RADC-TR-89-197
Final Technical Report
October 1989

AN INTELLIGENT TOOL FOR THE DESIGN OF PRESENTATIONS: A SYSTEM IDENTIFICATION STUDY

BBN Systems and Technologies Corporation

Bruce Papazian, R. Bruce Roberts, Donald J.A. Redick, Daniel M. Tani, Richard W. Pew, William S. Salter, Mark Friedell, Joseph Marks

APPROVED FOR PUBLIC RELESE; DISTRIBUTION UNLIMITED.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

RADC-TR-89-197 has been reviewed and is approved for publication.

APPROVED: Michael L. Mc Hale

MICHAEL L. Mc HALE
Project Engineer

APPROVED: Raymond P. Urtz Jr

RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER: Igor G. Plonisch

IGOR G. PLONISCH
Directorate of Plans & Programs

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

1b. RESTRICTIVE MARKINGS
N/A

2a. SECURITY CLASSIFICATION AUTHORITY
N/A

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE
N/A

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for public release; distribution unlimited.

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
6932

5. MONITORING ORGANIZATION REPORT NUMBER(S)
RADC-TR-89-197

6a. NAME OF PERFORMING ORGANIZATION
BBN Systems and Technologies Corporation

6b. OFFICE SYMBOL *(If applicable)*

7a. NAME OF MONITORING ORGANIZATION
Rome Air Development Center (COES)

6c. ADDRESS *(City, State, and ZIP Code)*
10 Moulton Street
Cambridge MA 02238

7b. ADDRESS *(City, State, and ZIP Code)*
Griffiss AFB NY 13441-5700

8a. NAME OF FUNDING/SPONSORING ORGANIZATION
Rome Air Development Center

8b. OFFICE SYMBOL *(If applicable)*
COES

9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
F30602-87-D-0093

8c. ADDRESS *(City, State, and ZIP Code)*
Griffiss AFB NY 13441-5700

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO
63728F	2532	QA	05

11. TITLE *(Include Security Classification)*
AN INTELLIGENT TOOL FOR THE DESIGN OF PRESENTATIONS: A SYSTEM IDENTIFICATION STUDY

12. PERSONAL AUTHOR(S) Bruce Papazian, R. Bruce Roberts, Donald J. A. Redick, Daniel M. Tani, Richard W. Pew, William S. Salter

13a. TYPE OF REPORT
Final

13b. TIME COVERED
FROM Mar 88 TO Oct 88

14. DATE OF REPORT *(Year, Month, Day)*
October 1989

15. PAGE COUNT
194

16 SUPPLEMENTARY NOTATION
Prepared in cooperation with Mark Friedell and Joseph Marks of Computer Corporation of America.

17. COSATI CODES

FIELD	GROUP	SUB-GROUP
12	05	
23	02	

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Intelligent Interfaces
Artificial Intelligence
Interface Design
Human Factors

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*
This report summarizes a six month effort to identify the appropriate role of knowledge-based technology in the design of user-computer interfaces and to assess the suitability of the published human factors design guidelines as a source of interface design knowledge. In addition to specific recommendations in these areas, the report incorporates a literature review of the following topics: the development of knowledge-based systems, rules-of-thumb concerning the types of problems to which knowledge-based systems have been successfully applied, the general nature of design problem solving, interface design methods, the use of human factors guidelines in interface design, and the state-of-the-art in interface design support tools.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL
Michael L. Mc Hale

22b. TELEPHONE *(Include Area Code)*
(315) 330-3564

22c. OFFICE SYMBOL
RADC (COES)

Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Table of Contents

Table of Contents-continued

List of Figures

List of Tables

Executive Summary

This report summarizes a six month effort to identify the appropriate role for knowledge-based technology in the design of user-computer interfaces and to assess the suitability of the published human factors design guidelines as a source of interface design knowledge. In addition to specific recommendations in these areas, the report incorporates a literature review of the following topics: the development of knowledge-based systems, rules-of-thumb concerning the types of problems to which knowledge-based systems have been successfully applied, the general nature of design problem solving, interface design methods, the use of human factors guidelines in interface design, and the state-of-the-art in interface design support tools.

In general, we have concluded that the use of knowledge-based technology to support design problem solving is still an active area of research. We have found that good interface design in particular is extremely dependent on the domain, task, application, and hardware environment, and we have concluded that knowledge-based systems appear to be most effective when applied to problems substantially more constrained than general interface design.

In evaluating the human factors literature and reports of attempts to incorporate the associated interface design guidelines into knowledge-based systems, we have found that these guidelines typically require further knowledge engineering before they can be used for this purpose, and that supplementary information concerning the specific application, task, and domain being supported must supplement their application. Furthermore, some of the most important emerging areas of interface design, such as the appropriate use of graphic display and direct manipulation techniques, have not been available for a sufficient period of time to have been covered fully by human factors guidelines.

As a consequence of the nature of the interface design problem and the characteristics of state-of-the-art knowledge-based technology, we recommend constraining the number of design parameters that the initial versions of a knowledge-based design tool must support before initiating a major development effort in this area. We recommend developing a modular, consultant based, mixed initiative architecture to provide an environment in which these constraints can easily be relaxed as design knowledge is acquired. We also recommend an initial program of knowledge engineering to support the development of domain-specific taxonomies of tasks and presentation primitives that can be used to solve interface design problems of immediate interest to the Air Force.

1. Introduction

1.1 Statement of the Problem

The current generation of Command and Control (C^2) systems can provide strategic and tactical decision makers with an unprecedented quantity of information. The way in which these systems present this information is often identified as a limitation of their overall effectiveness. While there is widespread interest in making the user interfaces to C^2 systems easier to use, and less costly and time consuming to build, a concensus on the best ways to achieve these objectives has yet to emerge (Andriole and Halpin, 1986).

One approach to improving the quality of C^2 user interfaces is to support their design with knowledge-based tools. Given the number of human engineering guidelines emerging for interface design, development, and evaluation, and the growing number of knowledge-based systems appearing in journals and conference proceedings that are designed to address this and other applied problems, it is not unreasonable to assume that knowledge-based systems that exploit human factors expertise could have an important role to play in solving interface design problems. Knowledge-based systems have captured the imagination of many, but one must keep in mind that the excitement associated with this new technology has led to extravagant claims concerning its maturity and suitability for application to ill-defined problems (Bobrow, Mittal, and Stefik 1986).

As knowledge-based systems are applied to a wider range of problems, experience is showing that they are no panacea and that they can be expensive to build. Like all technologies, they are best suited to particular types of problems, that is, those that are reasonably well-defined and bounded (Waterman 1986). Many of the most successful knowledge-based systems took several years to demonstrate a level of expertise that was much less general than expected. We do not mean to imply that the current generation of knowledge-based systems are too narrow to be useful or cost effective, but we want to stress that it is important to have realistic expectations of them and to carefully define the problem to which they will be applied before developing one.

There is little doubt that the costs associated with developing user interfaces are substantial. User-interfaces have been estimated to represent from thirty to fifty percent of the software making up typical interactive computer systems and interfaces are frequently described as one of the most difficult and time consuming aspects of these systems to develop (Bobrow, Mittal and Stefik 1986). How knowledge-based systems should be used to alleviate these problems is less obvious. In this report, the issues involved in answering this question are identified and recommendations for the application of knowledge-based tools to the problem of interface design are made.

1.2 Limitations of the Literature Review Approach to Knowledge-Based Technology Assessment

One way to advance our understanding of an appropriate role for AI technology in the interface design process is to conduct a literature review. A literature review is the right place to start. But before evaluating such a review, the reader should be aware of several issues that limit the value of this important preliminary step in the development of a knowledge-based system.

A literature review is the first of <u>many</u> steps that must be followed when designing a knowledge-based system and cannot be used as a substitute for a formal feasibility study involving systematic knowledge engineering. The repeated observation of experienced designers solving multiple design problems will still have to be done, along with the development of at least one prototype (see Section 2.1). Contrary to what many people have been led to believe, the development of expert or knowledge-based systems is still an area of artificial intelligence research, in which it is still difficult to judge fully the suitability of an application problem without experimenting with it first.

The most effective way to design user-interfaces is also an area for research. There is currently no widely accepted model of the interface design process against which to judge the value of existing systems or to define requirements for them. In fact, many of the guidelines found in the literature involve prescriptions for changing current practices that, in many ways, contradict one another. They range from increasing the formality of the requirements-definition phase of development to eliminating this phase due to the high level of effort associated with it and the marginal returns that it seems to yield.

Even though the human factors literature abounds with guidelines and recommendations for interface design, one must remember that they were not intended for machine interpretation. Most are written at a level that presumes they will be applied by professionals with considerable "common sense" and professional background expertise, making them appear very abstract to a knowledge engineer. Culling rules from the literature and "knowledge engineering" them into an executable form are substantial problems.

Another challenge in incorporating guidelines that promote better designs is the fact that there is a relatively little consensus concerning criteria on what constitutes a good interface design. About the only thing on which most human factors experts will agree is that to have a good design, the intended purpose of the system must be known. Our experience tells us that the consideration of this aspect of the problem is, in fact, a very important aspect of interface design that is often overlooked. Unfortunately, this implies that the interface design problem will be difficult to solve in general. Every new interface design problem probably will always require the acquisition of substantial knowledge of the user task that the system is intended to support.

Many reports we have reviewed describe interface design support systems that are in a very early stage of development. In spite of the broad claims of generality and problem solving power that their designers make for them, when many of the reports we reviewed were written, the systems being described typically are able to address only very restricted parts of the total design problem. The majority of the systems reviewed have not yet been used by interface designers for rigorous design-problem-solving evaluation.

1.3 Organization of the Report

In this report, the feasibility and practicality of applying knowledge-based technology to the user-interface design problem is discussed in terms of the state-of-the-art and practice of user-interface design and the state-of-the-art in knowledge-based system development. It begins with a discussion of the steps that were taken to develop several of the most successful large-scale knowledge-based systems. We also discuss the nature of the problems that they were successfully applied to, as well as the resources that were consumed during their development.

Following this discussion, we review the design literature, and present an idealized, prescriptive, human-factors oriented, interface design model intended to capture the design process, including the many external constraints that are typically considered when interfaces are developed. This prescriptive model is contrasted with another, more descriptive model, which we believe more accurately portrays the interface design process as it is commonly practiced. The two models are then discussed in terms of their suitability for knowledge-based automation.

Next, the results of a review of more than twenty interface design support systems are discussed in terms of the prescriptive design model presented in the previous section. We use the model to explain the state-of-the-art with respect to interface design tools. We do this to identify the areas of the design process that are not being addressed by the current generation of interface design support systems, and to identify areas that appear likely to benefit from the application of knowledge-based tools.

In Section 5, design proposals for an intelligent tool for interface design problems are presented in terms of the goals that are important for it to meet, the constraints that these goals impose on its architecture, its knowledge representation requirements, and desirable user interface characteristics. In addition to the literature discussed in the previous sections, the proposals in this section are driven by the requirements that are implied by what we feel would be an important capability for the system to exhibit. That is, it should be able to be used in a manner that is consistent with either the prescriptive or the descriptive models of the design process that we have identified.

This report concludes with two categories of recommendations for developing knowledge-based tools for the design of presentations; those that involve a low level of risk and largely exploit the current state-of-the-art, and those that would involve more risk and expense, but have the potential for advancing the state-of-the-art.

2. Knowledge-Based Systems Development

Because of the variety of forms that expert knowledge can take, many believe that knowledge-based systems can only be designed and built in conjunction with formal knowledge engineering activities that are focussed on the specific types of problems found in the field of study that the system will be asked to solve. In fact, many believe that it is only through this process that the requirements for the system's architecture can be defined, because it has to match so closely the structure of the knowledge that domain experts apply (Buchanan, Barstow, Bechtal, Bennett, Clancey, Kulikowski, Mitchell and Waterman 1983). One implication of this is that once a problem has been studied, it is possible that the available architectures, including hybrids, will not meet the requirements that a given domain places on it. Others believe that the expert system field has experimented with enough problems that "rules of thumb" concerning the size and nature of the problems for which knowledge-based systems are appropriate can be deduced (Bobrow, Mittal and Stefik 1986; Prerau 1985; Waterman 1986).

In this chapter we review a model of the knowledge-based system development process presented by Buchanan, Barstow, Bechtal, Bennett, Clancey, Kulikowski, Mitchell and Waterman (1983) as well as the "rules-of-thumb" offered by Bobrow, Mittal and Stefik (1986), Waterman (1986) and Prerau (1985). This development model and these rules-of-thumb provide a background for our discussion of the feasibility of applying knowledge-based tools to the design of user-interfaces for C^2 applications.

2.1 Incremental Development Model

The most effective way to develop knowledge-based systems is still an active area of artificial intelligence research. Contrary to what one might read in the popular press, most of the people building useful expert systems have studied artificial intelligence, experimented in knowledge engineering, and reworked prototypes for years before producing the successes like R1(Bachant and McDermitt 1984), MYCIN (Shortliffe 1976), and Dendral (Lindsay et al. 1980). It is fair to say, with the exception of some very narrowly focused and limited programs, that the process by which successful expert systems are produced is more accurately described as applied research than as development.

Expert system development is experimental in nature. Developers must conduct empirical research to determine how to solve problems requiring extensive knowledge. Buchanan, Barstow, Bechtal, Bennett, Clancey, Kulikowski, Mitchell and Waterman (1983) describe this process as evolutionary or incremental. In this section we summarize the process of taking a system from conception to a level of maturity that these authors call "adolescence." While this evolutionary process is described as a series of steps or stages that include identification, conceptualization, formalization, implementation, testing and revision, these stages are intended to be only rough characterizations of a complex and ill-structured design and development process that, in practice, varies from situation to situation. Our description closely follows the model presented in Figure 2.1.

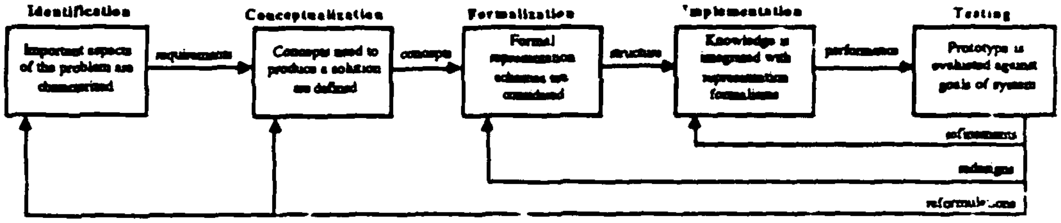

Identification
Important aspects of the problem are characterized
requirements
Conceptualization
Concepts used to produce a solution are defined
concepts
Formalization
Formal representation schemes are considered
structure
Implementation
Knowledge is integrated with representation formalisms
performance
Testing
Prototype is evaluated against goals of system
refinements
redesigns
reformulations

Figure 2.1 The knowledge-based system development process (adapted from Waterman 1986).

2.1.1 Identification

The first step in the development of knowledge-based systems involves identifying and characterizing the most important aspects of the application problem to be addressed. This can be started by reading about the domain and asking general questions about how things are done in general. Through these activities, hypotheses about how the knowledge of the domain might be structured can be generated and candidate architectures can be reviewed for ideas. We call this initial activity in the identification process "orientation to the domain."

The "real" knowledge engineering effort does not begin, however, until a knowledge engineer sits down with experienced people to learn how they approach specific problems. Ideally, this should involve many experts and many sessions in which knowledge is acquired, restated, and fed back for confirmation. During this stage, a rapport can be developed, and the expert(s) can be judged for their suitability in terms of experience, commitment, availability, and ability to communicate. The engineer gets to know, still in a relatively informal way, more specific information about the nature of the domain. The engineer might ask questions about what he has read and try to confirm some of his initial hypotheses, but, in general, the developer's role at this stage is best characterized as that of a student.

Once the developers have a reasonable understanding of the domain, they can begin to work with experts to clearly define the problem or subproblems that the system can be expected to solve. The selection of a candidate problem should be made as a function of the types of problems that are worth solving and the representation and reasoning schemes that are available to represent them.

Several iterations of this process are likely to be required before a problem is identified that is worth solving and can be described at a sufficient level of detail. In fact, at this stage of the development process, the ability to produce a sufficient level of description detail is one of the criteria for judging the suitability of a problem.

Once a general problem area has been defined, example problems and scenarios must be collected for study, including the support materials that will be required to simulate the normal solution process. The types of representation schemes that might be appropriate should be identified, a programming environment should be considered, and a budget/schedule should be initiated. At the same time, clear performance goals for the system should be made explicit, since they will affect these decisions as well. If the output of the system will be used by some other system, person or organization, the constraints that their application may place on the system output should be considered at this time as well.

2.1.2 Conceptualization

During conceptualization, the domain concepts and relationships defined during the identification stage are made more explicit. Buchanan et al. suggest making diagrams of these concepts and relationships so that they can be reused to support the parallel development of multiple initial prototypes. Developing a system conceptualization is a time consuming and difficult process that produces products that often must undergo revisions after a prototype is built. Diagramming concept relations usually requires repeated interactions with experts.

Attempts to apply formal implementation-orientated representations at this stage should be avoided because the best efforts to select appropriate formalisms for them can fail at this point. If they do fail, this step must be repeated for the alternative representations. Structured walkthroughs of specific problems that implicitly incorporate assumptions about representations are recommended however. Small prototypes might also be built, but the bulk of the collected data should remain stored in a neutral form to protect it against loss due to contamination from premature formalization.

2.1.3 Formalization

This is the stage where formal knowledge representation schemes are evaluated for mapping to the concepts, processes, and data that were modelled during conceptualization. One has to consider the granularity and structure of the relevant concepts and decide if they should be represented as some sort of organized system of objects or as sets of primitive entities. Developers must also determine if there are important causal or spatial-temporal relations among the concepts that must be represented explicitly, and decide if they can be related hierarchically or need to be represented at multiple levels of abstraction.

Defining the model or models of the underlying process that will be used to generate and evaluate solutions is another important part of formalizing the knowledge. This will require the consideration of techniques such as heuristic rules, hypothesis testing methods, formalized procedures, mathematical (analytic or statistical) transformations and a wide range of behavioral decision models (Stefik, Aikens, Balzer, Benoit, Birnbaum, Hayes-Roth, and Sacerdoti 1983). The nature of the data in the domain has to be understood and its treatment formalized as well.

Formalizing the conceptual information flow and relating it to subproblems can constitute a partial specification for building a prototype. This specification will follow from the choice of architectural framework and the explicit sketch of the concepts and their relations in the domain. Ideally, the selection of the development environment and tools will be driven by the particular nature of the conceptualizations and formalizations selected for each subproblem, but in some cases this choice will have been made already. Ideally, the developer will be able to work with software development tools that will minimize representational mismatchs and enable efficient development simultaneously.

2.1.4 Implementation

During implementation, the first executable programs are produced. They are used to test the local and global consistency of the representation schemes selected for the various knowledge and data types required to solve the general problem that the target system is intended to handle. The code produced at this stage is not necessarily intended to survive these tests. Some may be reused, but the primary objective of this process is to collect information on the adequacy of the products of the knowledge engineering work carried out in the previous stages and their compatibility with the implementation tools selected.

In a very real sense, requirements for the system are still being determined through this process. Unlike those defined in the previous stages, the major focus here is on the system's performance and on the design of the overall architecture. Many of the constraints that the final implementation will have to accommodate, such as those associated with the design and development of the user interface, are usually given lower priority than is ideal.

It is typical for initial prototypes to be constructed using whatever languages, structures, and high-level tools are available. When these tools are found to be inadequate, either they should be replaced, or new ones have to be built. It is not necessary to produce a comprehensive system at this point, but it is important to confront representative subproblems of the types that will eventually have to be confronted.

2.1.5 Testing and Evaluation

Once the implementation has progressed to the point where it can cover a complete problem, it needs to be tested to uncover its weaknesses by applying it to cases representative of those that it will have to support. The examples should be different from those that were used to structure the initial stages of the implementation, and the testing should involve representatives from the user community.

User-interface and I/O dialogue issues can now be studied in more detail. Data acquisition and the presentation of results can be evaluated as well as the basic issues of knowledge representation and manipulation. It is not uncommon for wrong or inadequate information to be solicited by early prototypes and for the language and style of the interface presentations to be difficult for users to understand. Issues of human engineering should be addressed, such as the degree to which the system offers its users a reasonable level of control and the extent to which it disrupts or is compatible with their natural sequence of problem solving activities.

The results that the system provides can be evaluated on a number of dimensions. They may be inaccurate due to flaws in processing, or due to inaccurate or insufficient knowledge bases. They may be presented at an inappropriate level of detail or they may omit information that the users will want to be able to access. The selected test cases can have a major effect on the evaluation process. Unless carefully chosen, the examples can be out of the intended scope of the system, but it is more often the case that they are too homogeneous. Buchanan et al. (1983) recommend explicitly organizing the examples to cover problems that are "difficult" and probe the limits of the system, problems that are "classical" or "prototypical" of the ones that it will face, and problems that exhibit a representative level of ambiguity or degree of missing data.

2.1.6 Prototype Revision

During incremental development of knowledge-based systems, revisions are expected to involve changes that can affect the concepts that are considered. the representations used, or the implementation. Prototype refinement normally involves revisiting the implementation and testing stages. When satisfactory performance is not obtained, more fundamental changes can be required of the architecture, knowledge bases, or the problem scope. Representations may require redesign and cause the development to revert back to the formalization stage. Difficulties that are even more serious can necessitate the re-identification of the concepts and processes used by the system or require the design to support the coverage of additional subproblems before adequate performance can be achieved.

2.2 The Selection of a Suitable Problem for Development

Like any other technology or technique, knowledge-based systems are not practical to develop for all problems. The current technology appears to work best when applied to particular types of well-structured and focussed problems, and, even then, requires a demanding development process that is difficult to circumvent. Proof-of-concept demonstrations can often be built in a few months time given the available rapid prototyping technology. This is true even when the most exotic problems are being attacked. However, if the intent is to build a large scale system that can approximate or augment the performance of an expert that must include knowledge spanning a reasonably wide domain, the time required to field it is more often on the order of years rather than months (Bobrow, Mittal and Stefik 1986). Table 2.1 summarizes the results of an informal survey of the scope and level of maturity of 176 expert systems reviewed by Waterman (1986). From this survey it is apparent that the majority of the systems were experimental at the time of Waterman's survey, and only a few have been exposed to use in the field.

2.2.1 Value of Solving the Problem

When making a decision to use knowledge-based technology to solve a problem, the first consideration should be economy. Simply stated, the problem must be worth solving, meaning that the savings realized must exceed the costs incurred. Unfortunately, given the inherently experimental knowledge-based system development process, it is difficult to estimate these costs accurately without first investing several man-years of time and effort studying the problem.

The stability of the domain is important to consider when making these labor projections. Some of the most efficient and practical knowledge-based systems are narrowly focussed, so that a changing problem may render a perfectly well developed system obsolete. Non-AI solutions that are not as interesting may, in fact, prove to be far more cost effective in the long run. For example, six months after the initiation of the development of DARN, a system designed to assist in the diagnosis and repair of computer disk systems, the circuit boards that had previously been difficult to repair became less expensive to replace than repair. This change made the original knowledge base obsolete, but more importantly, the need for the diagnostic aid was eliminated by this change (Mittal, Bobrow, and de Kleer 1985). On the other hand, Digital Equipment Corporation reports that the R1/XCON system can effectively configure VAX-11 systems based on customer requirements 97% of the time and repays its development costs every few months, but it took over 50 man-years of development work for it to reach this level of proficiency (Bobrow, Mittal and Stefik 1986).

Maturity Domain	Development Tool	Exporatory Testbed	Experimental System	Research System	Field Prototype	Production Prototype	Commercial Product	
Medicine		1	29	16	5	1	1	53
Military			13	10	1			25
Electronics			12	1	1		1	22
Chemistry		1	9	1	1	1	1	14
Computer Systems			3	2	2		3	10
Law		1	6	3				10
Information Management			1	7		1		9
Engineering			3	3	2			8
Geology			5		1	1		7
Space Technology	1		2	4				7
Mathematics			1	1			1	3
Manufact.			2		1			3
Physics			2					2
Process Control	1			1				2
Meteorology				1				1
	2	3	85	50	14	4	7	176

Table 2.1 Informal survey of 176 knowledge-based systems shows the majority to still be experimental (from Waterman 1986).

2.2.2 Availability of Test Cases

It is critical to have a representative set of problems available for study. They have to be equivalent to the kinds of problems that the user of the knowledge-based system will confront.

This includes the level of support information that can be expected to accompany them. These problem sets are necessary to structure knowledge acquisition sessions in which experts will be observed and interviewed concerning their approach to their solutions.

Difficulty in identifying test cases can in and of itself be very informative when determining the feasibility of applying knowledge-based technology to a problem. If they occur infrequently or are only anticipated, finding individuals with expertise in their solution can be a problem, and the costs associated with the development of a system to support this process may be difficult to justify. We have some experience in developing concept demonstrations for knowledge-based solutions to problems that were recognized as important but occurred rarely or were only anticipated. This suggests that the knowledge acquisition process can turn from an analytical problem into an exercise in prediction. We have found that the results of taking this approach are often so controversial within the application community that the best treatment of these predictions cannot compensate for the uncertainty associated with the quality of the contents of the knowledge-base.

2.2.3 Difficulty of the Task

At some level, the difficulty of the task that is suitable for a knowledge-based system will change as a function of the technology that is available, but other constraints that task difficulty imposes will remain constant. We may be able to search large solution spaces and develop new search techniques that enable us to attack "more difficult" problems from a computational point of view. But the time to "knowledge engineer" problems will probably remain some constant function of the time it takes experts to solve them.

Based on technology available five years ago, Buchanan et al. suggested that a suitable task is one that takes an expert no more than an hour or two, not counting the time spent on manual activities such as making sketches or filling out forms. In 1986, Bobrow et al. continued to support this rule-of-thumb because they feel that tasks taking longer than this are likely to be insufficiently bounded in terms of the knowledge that they require. They also state that tasks which take only a few minutes to complete are probably better suited to simpler technology.

Another issue to consider is what Bobrow et al. call the predominance of common sense knowledge; this can indicate a relatively unbounded amount of information is brought to bare on the solution. Eventually all this knowledge will have to be explicitly encoded, which can take much more effort than is practical to devote to it. At some level, common sense is plentiful and inexpensive to apply in its manual (or human) form, and as a consequence will probably cost more to engineer it into a system than it is worth.

In general, expert system technology appears to be suitable for automating tasks that perhaps are more routine and mundane than many have been led to believe. Problems that are known to require English language understanding, complicated geometric or spatial models, complex causal or temporal relations, or an understanding of human intentions are not seen as good candidates for the state-of-the-art knowledge-based technology (Bobrow, Mittal, and Stefik 1986).

2.2.4 Availability of Expertise

The development of a knowledge-based system requires the dedication and long term commitment of at least one expert (but ideally several) who is interested, articulate, and experienced. Knowledge engineering experience has shown that books and theories that prescribe how things should be done are suitable mostly for orientation to a domain but are of limited value when it comes to actual knowledge engineering.

A thorough explanation of the problems associated with using sources other than recognized experts is beyond the scope of this discussion. But in the course of attempting to acquire explicit models of problem solving processes from other less appropriate sources, it quickly becomes apparent that they are inappropriate for machine representation. In the best of situations, knowledge engineering is a difficult and time consuming part of the development process that is considered by many to represent the major bottleneck in the application and development of knowledge-based technology. Skilled problem solvers often find it difficult to explain how they use the knowledge that they have. It often appears that experts use specialized knowledge and heuristics to solve routine problems. They tend to reason from basic principles only when confronting the most rare and difficult cases and frequently are observed to use methods that appear to the uninitiated to be tricks and shortcuts when working in their field. It is the discovery and application of these methods that make the task of representing their problem solving process possible, but also difficult.

Attempts to take principles from books or other instructional materials have shown the knowledge they incorporate to be organized for the purposes of human communication, and to be cast in general enough terms to apply to many different specific problems, when augmented with common sense. The level of abstraction of these sources is typically too high to be applicable to any particular problem, and attempts to use them, in the past, have led to the development of systems that in concept could have been very general problem solvers, but in reality were never able to solve any problems in particular.

It is difficult to find experts that are suitable for the knowledge acquisition task. It is also difficult to find ones that can devote the time that it takes to iteratively acquire, represent, and test the knowledge that they routinely apply. They must be knowledgeable, willing, and able to communicate what they know. Ideally, they should have some sort of vested interest in the success of the project. The people that meet these requirements are seldom available. When they are, they tend to be very expensive. It is common for a sponsoring organization to offer the most expendable rather than the most expert individuals for these purposes. Furthermore, after the preliminary stages of knowledge acquisition, it is important to observe additional experts solving several different problems. Problems are typically solved using unique approaches. One of the challenges of the knowledge engineer is to structure the knowledge derived from people solving specific problems in such a way that it will be applicable to the range of cases that the system will be used to solve. If the solutions are radically different from problem to problem or across individuals, the domain is probably not a good application for knowledge-based technology. In terms of resource planning, this means that the cost of domain expert participation should be included as a considerable portion of the expense of knowledge-based system development efforts.

2.2.5 Clarity of Performance Criteria

The last characteristic to be evaluated when determining the suitability of a problem to apply knowledge-based technology to it is the clarity with which successful solutions can be defined. In general, the most successful applications of knowledge-based systems have involved problems that were large in terms of the number of details that must be considered, but tractable.

One way to identify a difficult problem is to question experts concerning the quality of a particular solution to a clearly stated problem. If there is little consensus on the value of a particular solution, it will be difficult to build useful measures of merit to control the system in its attempts to solve it. Furthermore, it will be difficult to justify the system. In some instances, it will be possible to identify performance criteria for subproblems that can then be used to define the value of an overall solution. However, there are many types of problems, such as the evaluation of designs and plans, in which sub-measures do not easily help to define overall merit (Stefik, Aikins, Balzer, Benoit, Birnbaum, Hayes-Roth, and Sacerdoti 1983).

Ill-defined problems are often the most important and certainly the most difficult types of problems that humans are asked to solve. They also can be some of the most interesting to work on and rewarding if, in fact, they are solved. However, they are quite difficult for computers to solve and are not likely to lead to the development of knowledge-based system that unequivicably will be considered successful.

3. Identification of the Interface Design Process

In chapter 2, an incremental design and development process for knowledge-based systems was reviewed, and a brief discussion of the nature of the types of problems for which knowledge-based systems appear to be most appropriate was presented. In general, the knowledge-based system design process calls for the early identification of the overall problem or subproblem that is to be solved by the system so that the available types of knowledge representation schemes, and the available formalisms can be assessed with respect to these needs of the domain.

In this chapter, we report our efforts to understand and model the user interface design process to support the identification of the areas of this design problem that appear most appropriate for the application of knowledge-based tools. In section 3.1, we review the published work that describes the processes of design in general, and what is known about the suitability of knowledge-based design-support tools for this domain in general. This is followed by a review of the literature that describes the interface design process in particular and the related fields of graphic design and statistical graphics. In Section 3.2, we summarize our findings with respect to the interface design process literature by presenting and discussing two development models, a top-down prescriptive model, and a bottom-up incremental development model. From the reviewed research, and our own design experience, we find that it is important to identify these two different development models and to seriously consider supporting the incremental development model, due to the fact that the prescriptive model which is representative of those advocated by human factors professionals and used as the basis for many of the guidelines they publish is frequently not followed by practicing designers. We conclude this chapter with a discussion of the need to consider both the prescriptive and the descriptive interface design process models when defining the requirements for the knowledge-based interface design support tools.

3.1 Literature Review

In this section, an informal review of selected literature relevant to the specification of interface design support tools is presented. We begin with a discussion of general theories of the design process and empirical observations of design practice, drawn largely from the knowledge-based design tool literature. We then focus on the human factors and graphic design literature that prescribes a design process for user interfaces, graphic presentations, and to a lesser extent, descriptions of the practice of user interface design as it occurs in the field. The section concludes with a discussion of the relevance of this literature to the design of user interface design tools.

3.1.1 Design Problem Solving

From the literature we have reviewed, it appears that the process of design is not well understood (Mostow 1984; Mostow 1985), yet it has attracted the attention of many artificial intelligence researchers (Duffy 1987) because of the large amounts of domain specific knowledge as well as considerable problem solving skills that it requires, which, on the surface, make it appear to be a problem suitable for the application of knowledge-based technology (Mittal, Dym, and Morjaria 1986). Many models of the design process have been produced to guide the development of knowledge-based design support tools that, as a whole, manifest a surprising degree of diversity in their assumptions concerning the process of design.

Brown and Chandrasekaran (1985), observed a hierarchical strategy of air cylinder design. Eastman (1968) described architectural design as a semi-hierarchical organization of "design units," and a serial process of problem identification, design unit manipulation, and constraint checking which contrasts with Gross and Fleisher's less structured model of architectural design which they describe in general terms as "the exploration of constraints" (Gross and Fleisher 1984). Cohen, May, and Pople (1986) found that the design expert they observed appeared to work from an evolving hypothesis about the solution to his problem. Adelson and Soloway (1984) characterized software design as the balanced exploration of a goal hierarchy, while Kant (1985) describes similar tasks in terms of a less balanced, non-hierarchical process. McDermott (1982) assumed a data-driven, bottom-up design model when he developed R1/XCON. Steinberg (1987), on the other hand, developed an initial version of VEXED, a circuit design tool, assuming a model of top-down refinement plus constraint propagation.

The range of apparently conflicting design models that have been constructed from the empirical observation of designers is probably indicative of two things. One is that the design process per say is a problem-dependent process and the second is that there are probably large differences in the ways individuals design within a given problem domain (Mittal and Dym 1985). For domains in which the design problem is well defined, the process appears to be consistent across experienced individuals (Carrol, Thomas, and Malhotra 1979; Carroll, Thomas, and Malhotra 1980; Carroll, Miller, Thomas and Friedman 1980; Mittal and Dym 1985) but is likely to vary as a function of design experience (Eberts, Lang, and Gabel 1987). In other domains, the process appears variable and to incorporate multiple approaches to design for even a single problem and for designers with equivalently substantial levels of experience. For example, both Steinberg and Shulman (1985), and Finegold (1984) have observed that VLSI design often has both top-down and bottom-up process advocates. This observation motivated Finegold (1984) to develop LOGICIAN, a commercial VLSI design environment, in such a way that either method could be followed, in spite of the fact that he personally was an advocate the top-down approach.

Overall, these findings suggest that a general, or universal model of design is not likely to be consistent with the process as it is practiced by people. Even if a general model of design could be identified, the architecture required to support it is likely to be too inefficient to be of any practical use (Bobrow,Mittal and Stefik 1986). Instead, it appears that a more practical approach to supporting design is not to attempt to automate the entire process. It appears more reasonable to consider subproblems that that can be handled effectively and efficiently, given today's technology. The process of narrowing a broad problem area to a size that can be managed by knowledge-based technology is what Waterman (1986) calls the "scoping process." This can be accomplished by focussing on a particular type or category of design, or some aspect of the overall design process, or perhaps, the design of a particular product. While categories such as routine versus creative design (Mostow 1985), or conceptual versus rational design (Chignell, Cho!, Petersen, and Miller 1987) have been discussed in the literature, research leading to the development of a taxonomy of design problem types is just beginning.

Peterson, Nadler, and Chignell (1987), for example, have suggested that design problems be organized into classes such as: invention - when an entirely new product is conceived and developed, innovation - when an existing product is adapted to a serve a new function, improvement - when an existing product is modified to satisfy broader or more stringent performance criteria, and refinement - when an existing product is redesigned to allow it to be developed using new methods or materials while maintaining a consistent purpose or level of performance. They point out that prescriptive strategies for design often describe either what should occur during the process of design but not how it should be carried out, or they define the characteristics that a product should exhibit without linking them to the process that would enable their realization. They take the position that the best way to approach a design problem is conditioned by the nature of design problem encountered. They recognize that a design-problem, design-process mapping is currently beyond the state-of-the-art in design theory, but they argue that these factors have to be accounted for when prescribing design methods and requirements for design aids.

Tong (1987) has also taken a domain independent point of view to develop a classification scheme for categorizing existing knowledge-based design models at the knowledge, function, and program level. Tong's work has a computational orientation, and like the Peterson, et al. taxonomy, its predictive power remains to be tested.

3.1.2 The Design of User-Computer Interfaces

The design of user-computer interfaces requires large amounts of domain-specific knowledge and problem solving skills. Because of the range of problems that computers can be used to solve, and the rapid development of new interface technologies, interface design has become a challenge that is more difficult than ever before. Interface design, like most areas of design, is a process that is difficult to describe in terms that are (1) precise enough to be useful to people doing specific work, and (2) general enough to remain relevant for any reasonable period of time, or for more than one specific application.

Like the written design guidance provided for other domains, it is common for interface design information to be published in a more fragmented form than many believe is ideal (Maguire 1982). It typically is presented in terms of: a general philosophy, principles or goals (see for example, Norman, 1986); a prescriptive model for the design process (for example, Williges, Williges, and Elkerton 1987; Boar 1984); guidelines and criteria for assessing the quality of good designs (for example, Smith, and Mosier 1984); or the results of evaluating alternative solutions to specific design problems (for example, Bly, and Rosenberg 1986).

Currently, the most popular philosophy of interface design is that of "the user-centered approach" (Norman and Draper 1986). The key principles of this philosophy involve the consideration of the system users' capabilities, tasks, goals and information requirements early in the process of a system's design. Its basic structure comes from "the systems engineering approach" to design and development. Many forms of this approach have been presented (Meister, 1971). In general, it is an elaboration of the frequently-advocated method of problem solving consisting of the definition of a problem, the development of alternative solutions, the testing and selection of an alternative, followed by an implementation.

Within this general model, the user-centered design philosophy calls for explicit resources to be applied to the definition, design and evaluation of the user interface of interactive systems. Special consideration is given to modelling the anticipated problem solving processes of the target users prior to the development of the system interface, through a task analysis (see for example, Smith, Irby, Kimbal, and Verplank 1982). In many respects, the user-centered design philosophy has been advocated by human factors professionals, who traditionally are responsible for insuring that person-machine systems are not designed with unrealistic assumptions concerning human performance capabilities built into them, throughout the history of their discipline (Meister 1985).

Many prescriptive models of interface design have been published (Williges, Williges, and Elkerton 1987; Rubenstein and Hersh 1984; Rouse 1984). Using the terms of the knowledge-based design literature, they generally prescribe an iterative top-down refinement process with the application of guidelines or constraints, and evaluation based backtracking. One of the most thorough interface design process models has been presented by Williges, Williges, and Elkerton (1987). Their three stage model is shown in Figure 3.1, with additional references for each step. Like most of the interface design process models, it has a strong sequential component to it, and it is presented with caveats concerning the recognition, that in practice, it will change as a function of the particular application and situation, but should include at least two generate and test cycles.

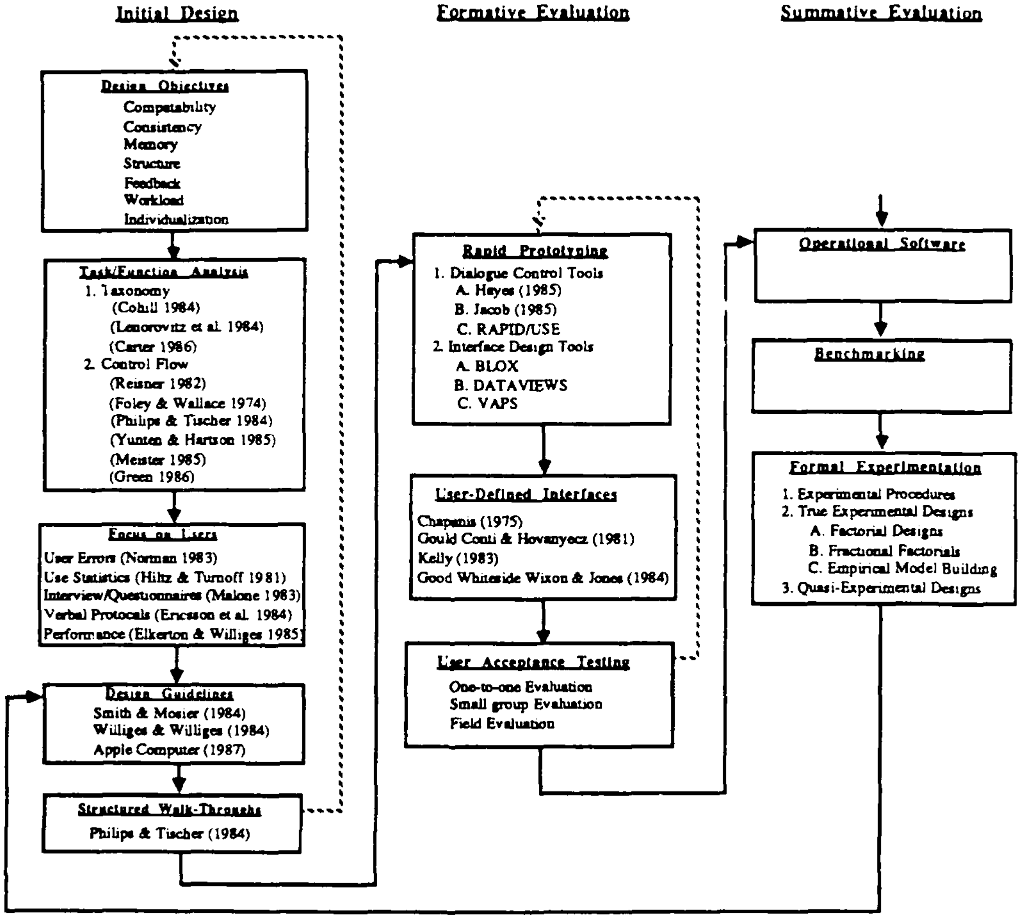


Figure 3.1 Prescriptive interface design process (adapted from Williges, Williges, and Elkerton 1987).

The most important observation to be made concerning this model is that Williges, Williges, and Elkerton (1987) recommend not only the specification of detailed requirements, a task and dialogue analysis as well as attention to traditional control/display and interface design guidelines. They indicate the need for user involvement at two points during the development process, one in which the design is evaluated for its adequacy, called the formative evaluation, and a summative evaluation to insure that the implementation conforms to the design. The fact that these recommendations are made, indicates the uncertainty associated with making adequate interface designs specifications without making refinements based on empirical observations of their results. The lack of adequate evaluation models is another reason for some of this uncertainty. Even the most aggressive prescriptive design model reviewed (Rubenstein and Hersh 1984), included at least one evaluation or "generate and test" cycle in the design plan.

Prescriptive models like those of Williges, Williges, and Elkerton (1987), and Rubenstein and Hersh (1984), are idealizations of the design process that do not relate well to how software interfaces design is practiced. Software development projects are usually managed by engineers without a background in human engineering. As a consequence, they often have a somewhat limited understanding of the organizational requirements that this work requires for it to be done effectively, or the need for it to be done at all (Gould and Lewis 1983; Perrow 1983). As a rule, software development projects do not incorporate the extensive analysis and test schedules that prescriptive interface design models advocate (Mantei and Teorey 1988), so that when human engineering problems arise, often there is insufficient time to carry out the background work related to task, dialogue, and domain analyses that could lead to more effective solutions (Meister 1987). When compared to other engineering disciplines, human engineering is often done in an environment that is more time constrained and resource deficient (Meister 1987). This makes the prescriptive design models advocated for interface design that do not account for these constraints even more unrealistic than prescriptive design models are judged to be in general (Nadler and Peterson 1987; Peterson, Nadler, and Chignell 1987).

The Williges et al. model, is an example of a nominal, prescriptive process for interface design. Compared to the number of descriptive design process models developed to support the design of knowledge-based systems for other domains, there have been relative few descriptive interface design process models reported. Nevertheless, like most descriptive design models, they describe processes that differ from what is typically prescribed (Meister 1987). For example, Hammond, Jorgensen, McLean, Barnard and Long (1983) interviewed five designers to produce the process model shown in Figure 3.2. When comparing the design practice described to them with typical prescriptive models, Hammond et al. (1983) noted that designers expressed a desire to have more information about user and task characteristics when making design recommendations, than they typically have, which could be explained by deviations from the prescribed model. But the designers also described historical, organizational, application, and personality constraints as other factors that have major effects on the process, which most prescriptive interface design models ignore. They described their informal methods to understand target user tasks, such as looking at newspapers to select appropriate chart graphics, or considering informal theories of human behavior as substitutes for empirical user analyses. Most felt that the human factors guidelines with which they were familiar, were typically too narrow to bother with or they addressed aspects of the design problem that were irrelevant.

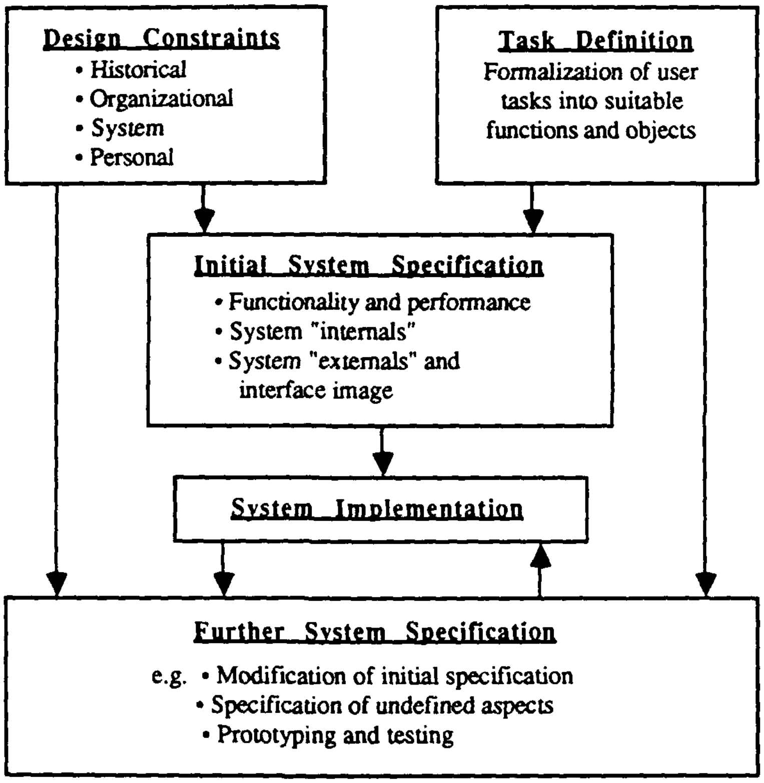

Design Constraints
• Historical
• Organizational
• System
• Personal
Task Definition
Formalization of user tasks into suitable functions and objects
Initial System Specification
• Functionality and performance
• System "internals"
• System "externals" and interface image
System Implementation
Further System Specification
e.g. • Modification of initial specification
• Specification of undefined aspects
• Prototyping and testing

Figure 3.2 Descriptive interface design process (adapted from Hammond et al. 1983).

In another study (Rosson, Mass, and Kellogg 1987), an informally selected sample of twenty-one designers were asked to discuss several specific design efforts they had been involved with. Rosson et al. report that the sample was evenly split with respect to how they designed interfaces. Half reported an incremental development process, in which design and implementation occurred simultaneously and in a highly iterative fashion. The others reported a phased process, in which there was an iterative design stage that incorporated some prototyping, followed by implementation. The particular process used appeared to be related to the business status of the project, such that the incremental approach was taken for projects with a research-orientation, while the more formal approach was taken when the product was scheduled for internal or external release. Virtually all the designers expressed the opinion that an appropriate design requires a comprehensive understanding of the task domain and target users. However, contrary to common prescriptive models, neither process appeared to include user involvement until after a functioning system was available. In both cases, the functional characteristics of the underlying applications were seen as major factors in determining the overall form of the interface.

One of the most well-balanced and successful system development processes described in the literature, and the one that most closely follows the user-centered design philosophy was that of the Apple Lisa computer (Morgan, Williams, and Lemmons 1983; Williams 1983). In fact, the process has been described as remarkably innovative because, "the designers did what designers are supposed to do; define the product's prospective customers, determine their needs, and then design a product to meet those needs" (Williams 1983). Usability objectives were central to the project goals and were made explicit to the development team through a standards document that they developed jointly over a six month period before starting. Hardware was developed to support, rather than dominate, the usability goals, and systematic and repeated user testing was used to revise the standards document, the design of both the hardware and software, and the functional objectives of the project.

But unlike most software development efforts, the interface was the primary focus and an unprecedented amount of time and resources were invested to meet predefined usability objectives. The desk-top metaphor was derived from the Xerox Star operating system, which involved 30 man years of design work before any software was written (Smith, Irby, Kimbal, Verplank and Harslem 1982), followed by investment of an estimated 200 additional man years of effort, using staff with several years of prior experience. This process deviates from typical prescriptive models but is consistent with descriptive models in terms of the extent to which hardware constraints were considered a part of the design. However, unlike most descriptions of real world interface development efforts, the hardware constraints were relaxed to support usability objectives rather than the converse.

Several interface design guideline documents have been published that tailor general human engineering design principles to specific aspects of computer interface design. The most comprehensive set is provided by Smith and Mosier (1984), followed by Williges and Williges (1984), Galitz (1985), and the guidelines for the Apple Desktop Interface (Apple Computer 1987). The Williges and Williges (1984) document is broad but lacks extensive indexing or cross references facilities, Smith and Mosier (1984) is very detailed, has a good index, and is frequently cited as one of the best documents of its kind, while Galitz (1985) focuses exclusively on screen based presentations for alphanumeric applications. The Apple guidelines (Apple Computer 1987), are broad in philosophy, and they cover direct-manipulation icon-based design, but they are oriented exclusively toward insuring that developers of Apple software applications use the Macintosh interface utilities consistently.

Due to the short time since graphic direct-manipulation interfaces have become a viable technology, the majority of the guidelines presented in these documents are, at best, appropriate for interfaces that are primarily alphanumeric. They assume at least passing familiarity with general human engineering design principles, and when applied to specific design problems, are often considered by designers to be either too general, or too specific (Maguire 1984). This is a criticism that has been leveled at human factors design recommendations many times in the past (Rogers and Armstrong 1977; Chapanis 1970), which we believe, in part, is due to the contextual and task-specific nature of what constitutes good interface design, and to the tendency of researchers to obscure the relatively limited contexts of the often well-controlled experimental studies which support the guidelines they present. Many guideline documents, on close inspection, appear contradictory. This, Maguire (1984) interprets as evidence that they represent alternative strategies, which are more or less appropriate, depending on specific, but often unspecified circumstances.

Early research concerning the most effective ways to present human engineering data for design purposes indicates that information phrased in quantitative, graphic, or tabular forms is preferred over qualitative or verbal presentations (Meister and Farr 1967), but that most designers prefer to rely on their own experiences, or the advice of other designers rather than on handbooks, guidelines, or standards (Rogers and Armstrong 1977). Most designers appear to think of human engineering issues as design constraints rather than as design objectives and to be indifferent to abstract, general design principles and guidelines (Meister 1987). They appear to prefer human engineering information that relates to specific "design problems," and system variables which does not characterize most of the written Human Engineering guidelines (Meister 1987). They would like to see this information in a form that would help them resolve design conflicts rather than to set design objectives. Another major criticism leveled at human enginering design documents is that specific design "actions" typically are not described and have to be inferred by the reader from general statements. The actions that the reader must perform appear to be assumed to be obvious by the design document authors.

Meister (1984) recommends that each guideline in documents such as Smith and Mosier (1984) ideally should be supported by information of the following type:

- A description of the behavioral requirements to be satisfied by a the design;
- Alternate ways of meeting the requirement, that is, alternate design solutions to the design problem;
- Limits within which the design feature will function;
- Expected performance from using each design alternative;
- The relative advantages of incorporating the design features offered;
- Cost associated with failing to incorporate a feature;
- Empirical evidence to support cost-benefit tradeoffs.

Maguire (1984) makes similar recommendations with respect to presenting alternative ways of meeting a oesign objective. The extensive research required to produce such guidelines has not been carried out, nor is it clear that it is reasonable to expect that it ever will be, given the range of factors, situations, and solutions involved, as well as the rates at which the availability of alternative solutions and their associated costs change. Such ideal design guidelines currently do not exist.

Williges, Williges and Elkerton (1987) believe that any data base, that included a comprehensive set of guidelines such as those proposed by Meister (1984) would probably become unmanageable in either guidebook or on-line form. Consequently, they have recommended the exploration of rule-based systems to manage this complexity. Two proof-of-concept versions of such systems have been built . Lenorovitz and Reaux (1986) have developed a prototype system using an EMYCIN-like goal-directed, backward-chaining architecture that runs on an IBM PC/XT. Its knowledge base was developed using selected *Smith and Mosier guidelines* (1984). They found the guidelines inadequate for their purposes without modifications and expansions accomplished through additional knowledge-engineering activity using experienced designers. The guidelines and the knowledge engineering activity combined produced a complex 180 - rule set for advising novice interface designers on mechanisms for choosing or entering text processing commands in text-manipulation applications.

The developers reported that the generation of this knowledge base with an obviously limited degree of coverage proved to be a much more difficult problem than they had anticipated it would be, which raised questions about justifying the effort required for any significant expansion of the system's coverage. They observed that the size of the rule-set appeared to increase geometrically with what seemed to be only minor increases in problem coverage and they expressed concern about the degree of problem-defining data that users had to enter into the system before it could make design recommendations.

In a more ambitious effort to leverage human factors guidelines, Frey and Wiederholt (1986) have attempted to develop a Display Design Expert System, for the aircraft cockpit domain, to compensate for the deficiencies in the current format of the available guideline documents by, "Providing valuable information to the designer in the appropriate form at the appropriate time," as part of a knowledge-based display design environment (KADD) that also included an information requirements data-base, a display editor, and a display simulation driver.

The system includes display design guidelines in a production rule format, for consistency checking, information requirements checking and the provision of human factors guidance. It also includes a current display knowledge-base, an information requirements knowledge-base, and an inference engine for relating requirements and design rules to the current display. The human factors rules it embodies relate to symbol size, number of colors, contrast, layout, and display primitive usage, and they appear to be relatively simple (for example, if measurements are exact then use digital displays, if measurements are approximate or relative then use scales with pointers).

Frey and Wiederholt (1986) reported one of the most significant difficulties in developing KADD to be that the human factors guidance on display design is based on metrics that are, "In a different dimension than the display characteristics that designers manipulate," and that the guidance is presented at too detailed a level to evaluate "holistic issues" of presentation design. For example, they report that their experience indicated that designers think of display types, precisions and special locations while the human factors guidar.ce is oriented toward issues such as symbol size, coding techniques, blink rates, etc.

The specific sources of information that Frey and Weiderholt attempted to exploit were not reported. However, in the past Chapanis (1970) has leveled similar criticisms at human engineering research in general. He has observed that the scientists who carry out the majority of human engineering research, do this work in academic settings, where there the incentives are greater to study variables that will yield statistically significant experimental results than they are to study variables that relate to the sorts of system design problems that engineers confront.

3.1.3 Graphic Design

One of the oldest applications of basic research in human perception to man-machine design is that of control and display design (Helander 1987). However, the recent development of hardware/software capabilities that can support the use of dynamic, high resolution, icon-based, bit-mapped graphic displays has occurred much more rapidly than the development of human engineering guidelines for their use in user interface design applications (Green, Sime, and Fitter 1981; Helander 1987). While some literature devoted to these subjects is available (Green and Pew 1978; Kaman 1975; Swanston and Walley 1984; Wainer and Thissen 1981) and while most of the traditional human engineering display design literature is, in fact, relevant to at least some aspects of the problem, many developers eager to apply the new display technology are attempting to fill the voids in the human performance literature by consulting graphic design publications for guidance in the appropriate use of charts, graphs, diagrams, maps, pictorial and icon-based displays.

In 1985, Kosslyn (1985) reviewed five books on the construction of charts and graphs (Bertin 1983; Chambers, Cleveland, Kleiner, and Tukey 1983; Fisher 1982; Schmid 1983; and Tufte 1983), and evaluated how well the recommendations offered by each book work within the constraints of what is known about human visual perception. Table 3.1 summarizes his evaluations, in terms of a visual information processing model Kosslyn has developed.

Based on this review, Kosslyn concludes that these books provide valuable insights into the design and use of graphs, but that in some cases the advice they give is not well founded. His opinion is that they all could have profited from a consideration of available facts about how humans process visual information, but that additional research was needed to support some of the books' recommendations. He states that much of good display design is still an art, and that it probably will be some time before a computer can emulate the "intuitions and wisdom" in these books. In his closing remarks he recommends building a knowledge-based system to emulate an expert graphic designer not because he believes we know enough about this area to be able to produce a useful system, but because he believes it would serve as a useful means for systematically defining the research questions that must be addressed to make graphic design more scientific (see Mackinlay 1986).

Graphics Book Criterion	Bertin	Chambers	Fisher	Schmid	Tufte
Readability	4	9	9	8	10
Generality	10	4	4	8	7
Discriminability	8	2	7	3	6
Visual Properties	9	7	7	5	7
Processing Priorities	7	6	3	5	5
Perceptual Distortion	7	9	5	5	7
Perceptual Grouping	8	7	5	5	2
Memory Limits	9	3	5	4	7
Ambiguity	9	5	8	7	6
Inferences	7	8	7	6	8
Purposes	9	4	4	5	4
Questions	8	8	6	5	3
Data Formats	10	9	8	7	7

Note: 1 indicates very poor; 10 indicates excellent

Table 3.1 Summary of graphic design book evaluations (adapted from Kosslyn 1985).

Statistical graphics is another area of graphic design research that considers computational and perceptual factors in graphic design from a data analysis point of view (Cleveland 1987). Experimental work to develop a science of graphics has recently been initiated at Bell Laboratories and elsewhere (Cleveland and McGill 1984; Simkin and Hastie 1987; Follettie 1986; Pinker 1981, 1983).

Cleveland and McGill are statisticians developing a theory of graphical perception designed specifically to overcome the ad hoc nature of most graphic design guidelines, like those reviewed by Kosslyn. Since 1983 they have been reviewing the graphics and psychophysics literatures, and conducting empirical research to develop an applied theory of graphic perception (Cleveland, Harris and McGill 1983). The basic premise of their theory is that there are 10 elementary coding schemes that people must interpret when extracting information from graphs, and that the judgements the different schemes require can be rank ordered in terms of the accuracy with which they can be made (Cleveland and McGill 1984) (see Table 3.2).

Rank	Coding Scheme
1	Position along a common scale
2	Position along nonaligned scales
3	Length
4	Angle
5	Slope
6	Area
7	Volume
8	Density (amount of black)
9	Color saturation
10	Color hue

Table 3.2 Graphic coding schemes ranked by ease of perception (adapted from Cleveland and McGill 1985).

They propose the use of this rank-ordered list when deciding how best to display data and also how best to trade-off aspects of display real estate when compromises must be made. Several preliminary experiments designed to test aspects of this theory have demonstrated its value for making pragmatic graphic display decisions (Cleveland and McGill 1985,1986; Dunn 1988). There has been no formal experimental verification of the codes of rank 8-10. Consequently, their ranking is still conjectural, but informal evidence places density and color below the other codes.

The same researchers propose a theory of slope perception to guide the specification of the shape parameter of two variable graphs (Cleveland, McGill, and McGill 1988). In essence, they have demonstrated that a heuristic that brings the slope of a plotted line closer to plus or minus 45^{o} will enhance slope judgements.

Simkin and Hastie are less optimistic about the robustness of Cleveland and McGills' theory (Simkin and Hastie 1987). They looked at the Cleveland and McGill theory in the context of work done by Follettie (1986) and Pinker (1981), which suggests that the relative effectiveness of the coding schemes will interact with the analytic task that the observer is undertaking, and that experienced users are likely to have "graph schemas" in their long term memory, which will interact with the relative ranks of Cleveland and McGill's prioritzed coding schemes. These findings, while preliminary, lead Simkin and Hastie to conclude that proposals concerning the elementary processes involved in graph perception should be taken only as serious speculation at this time.

Further experimental evaluation of the effects of these additional task and domain factors will have to be explored before the robustness of simple performance rankings such as those offered by Cleveland and McGill can be determined. Simkin and Hastie (1987) project that taxonomies of the analytic tasks and the judgements these schemes are expected to support will eventually have to be developed and integrated into recommendations for the selection of graphing methods before they can be used with any degree of confidence.

Researchers at Bell Laboratories have also been experimenting with dynamic graphics for data analysis that allow real-time, direct manipulation of computer displayed presentations (Becker, Cleveland, Wilks 1987). The work being done in this area is exploratory and largely atheoretical at this point. A collection of papers on this topic is expected to be published in 1988 (Cleveland and McGill in press).

3.1.4 Relevance of Literature to Expert System Design

In preparing to identify a suitable model of the interface design process, we have reviewed the following: literature describing the process of design in general, models of the design process developed to support the design of knowledge-based systems for domains other than user interface design, prescriptive guidelines and descriptive models of the interface design process, graphic design books, and the statistical graphics design literature. In this section several conclusions are drawn concerning this literature and its relevance to the specification of knowledge-based tools for interface design.

First of all, we have observed that the design process involves a broad range of problem solving methods that for many domains are not well understood. Design methods have been shown to vary from individual to individual, from domain to domain, and to depend on the way the initial design problem is defined. This observation has led some researchers to conclude that "design," per say, is a problem area too broad and domain specific to be able to be supported by a universal knowledge-based design architecture. The need to develop a design problem taxonomy that will structure the apparent contradictions in the design process literature has been identified in the literature, and work in this area has started.

As knowledge engineers working in other domains have found, textbook descriptions of problem solving processes rarely correspond to how it is carried out in the real world. Design problem solving in general, and interface design in particular do not appear to be exceptions to this rule-of-thumb. Experts from all domains appear to develop relatively shallow, but situation specific, heuristics after years of experience that guide their problem solving process, and they appear to resort to "first principles," and guidebooks for only the most exceptional cases.

Prescriptive user-interface design models tend to describe a process that assumes the application of more resources to the definition of user requirements than is typically applied, and they tend to omit guidance for making tradeoffs with the other factors that are involved in designing interactive systems. These factors include conflicting management priorities, time and cost factors, product history, and other hardware/software constraints. They also appear to assume the type of design that Peterson, Nadler, and Chignell (1987) call invention, when in fact the majority of design problems interface designers appear to address, even when new underlying applications are being developed, is innovation or improvement.

In practice, design guideline documents are frequently ignored because of the generality with which they make recommendations, the extent to which they do not include the myriad of contingencies that affect the relevance of specific factors, the extent to which they are found to be internally inconsistent, the extent that they are based on idealized or inappropriate design process models and priorities, the extent to which they deal with variables that designers do not typically manipulate during design, and the difficulty designers have locating information relevant to specific design issues.

Recommendations for improving guidelines have ranged from incorporating all the trade-off and contingency information that designers would like to have, to developing rule-based systems for selecting appropriate advice. Limited experience with the latter recommendation has suggested that technology to support such an application is currently available, but that the current guidelines would have to be substantially augmented for them to serve as an adequate knowledge base. Also, the task and domain specific nature of the interface design problem requires so much context-defining information before rules can be applied that the value of taking this approach has been called into question by those who have experimented with it.

The biggest problem experienced designers report when specifying user interfaces is not that they do not know how to satisfy user requirements once the domains and tasks are understood. The problem is that they rarely are able to collect this information due to scheduling constraints that limit access to representative target users at critical times of the development process. They often also lack sufficient resources to carry out sufficient user analyses, or are not able to use the technology that would be most effective for the problems they have identified because of the requirement to reuse already available technology in an effort to same time or control costs. The design recommendations that human factors can make are often not incorporated into a design for a variety of other reasons, such as, management misunderstanding, or the costs associated making changes to other parts of the system if attempts were made to bring the design into compliance.

We have learned that translating guidelines into rules is neither a straight forward approach to difficult design problems, nor is it an appropriate approach to take and we believe that developing a design support system based on prescriptive interface design models that are not frequently followed will result in a system that will not be used. The issues involved in improving the usability of user interfaces are complex, requiring creative solutions that will not necessarily come from text-books or guidelines. Solutions will require careful consideration of the designer's problem in its broadest terms as well as a detailed evaluation of the state-of-the-art with respect to design support tools. In section 3.2 we present a model of what we believe to be the most optimistic process that interface design can be expected to take placed in, as well as a description of a process that we feel is more characteristic of how most interfaces are produced. In Chapter 4, we review more literature describing the functional capabilities of tools that are currently being marketed and researched to facilitate the interface design process.

3.2 Best- and Worse-Case Interface Design Process Models

Prescriptive models for interface design are usually skewed to emphasize the issues that are important to the discipline reporting them, and human factors models are no exception. They tend to present the process in ideal terms with unrealistic assumptions and little consideration for the myriad of factors that influence the process beyond the technical issues of interest to the discipline (Hammond, Jorgensen, McLean, Barnard and Long 1983).

In this section, two contrasting interface design processes are presented, a best- and a worst- case model, to offer a broader perspective on the user interface design process than the simple prescriptions that can be drawn from the human factors literature. The emphasis for both cases is on characterizing the number of factors and types of knowledge that are involved in addition to human engineering and the ways in which the design process may change w.ien "innovation" rather than "invention" is taking place (Peterson, Nadler, and Chignell 1987). Individually, the models illustrate the multidisciplinary nature of the interface design process. Together they demonstrate different ways the same bodies of ideas might be applied to the same fundamental design problem as a consequence of different priorities and assumptions concerning the best ways to approach the development of interactive systems.

3.2.1 Best-Case Model

The best-case model is described in terms of: task model development, allocation of functions, dialogue model development, assignment of interface primitives, development of screen presentations, design of presentation transition methods, and prototype construction. We call this a best-case model, because we have assumed the primary goal of the overall system design effort to be usability, that sufficient resources will be available to meet that end and that backtracking is permissible from any point in the design process to any previous point. These are perhaps the best conditions under which interfaces are designed and there existence seems to be an implicit assumption that underlies many of the available interface design guidelines.

This model is too abstract to be used as the knowledge-base for an interface design tool As we discussed in chapter one, a model sufficient to guide a design effort would not come from a literature survey alone, but would require the study of experienced designers working on representative design problems to be solved by the system. This model is offered as a hypothetical abstract description of what the outcome of such a knowledge engineering effort might look like, in terms of the types of knowledge, activities and sequences of actions that would be involved, if the design process being observed was accomplished under the best of circumstances (see Figure 3.3).

3.2.1.1 Task Model Construction

In many ways, the knowledge acquisition requirements for developing a user interface are as demanding as the requirements for developing an expert system (Betts, Burlingame, Fischer. Foley, Green, Kasik, Kerr, Olsen, and Thomas 1987). A conceptual model of the users' problem solving task has to be constructed to understand the sequences of activities involved, as well as the goals, concepts, language, and information that is involved (Hollanagel and Woods 1983). While this kind of knowledge acquisition activity is relatively new to computer science (Card, Moran, and Newell 1983; Norman 1986), human factors and industrial engineers have been collecting this type of information for years to support the development of large scale interactive systems using a variety of task, function and operational sequencing analysis techniques specifically developed for this purpose (Drury, Paramore, Van Cott, Grey, and Corlett, 1987; Meister 1985).

To produce an overall task model in the form of tables or graphs that capture the range of methods and individual differences likely to occur during normal problem solving, several users of the system must be interviewed and observed in their natural context while they are using typical support tools or materials. The modelling activity is usually iterative, requiring

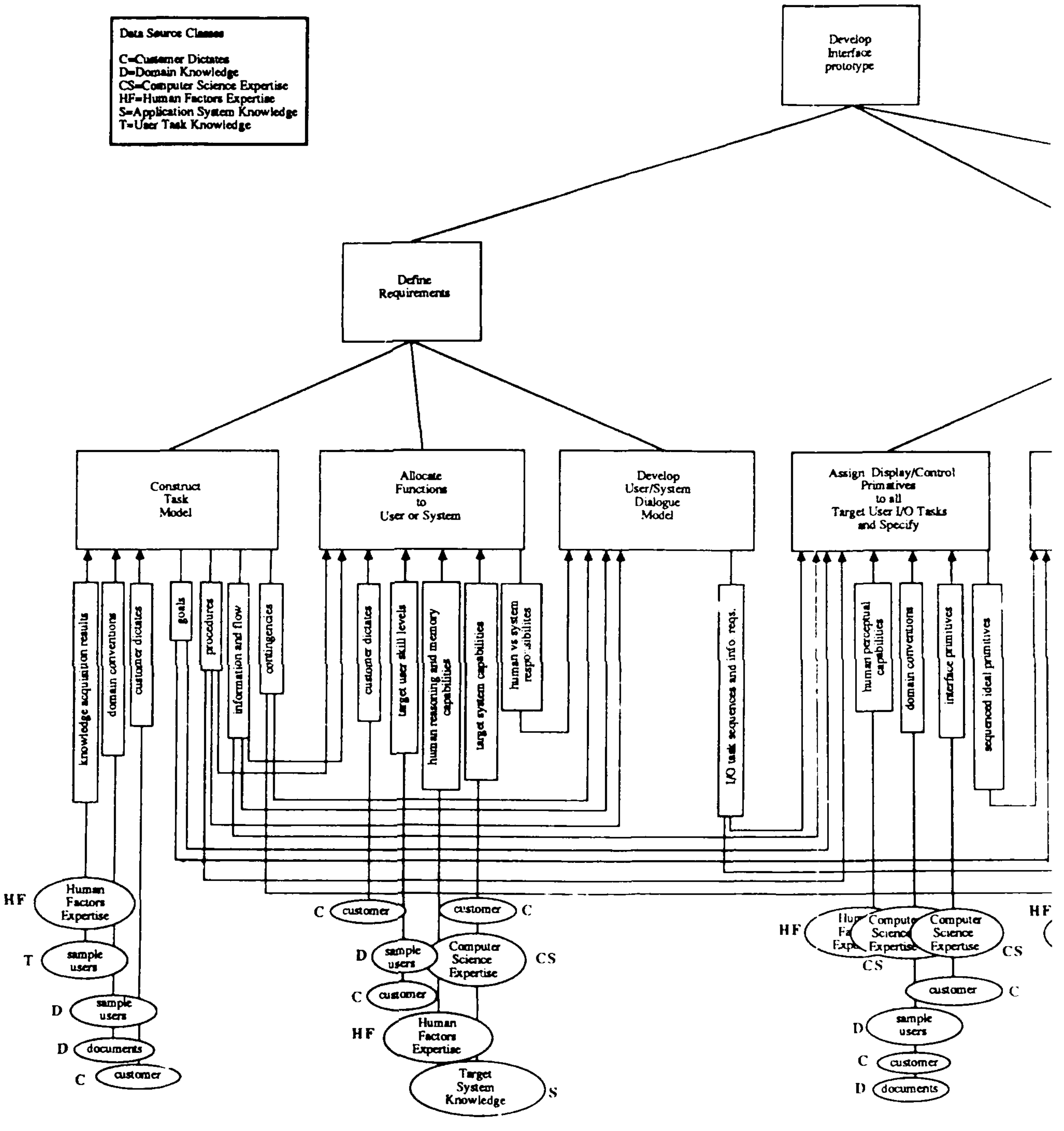
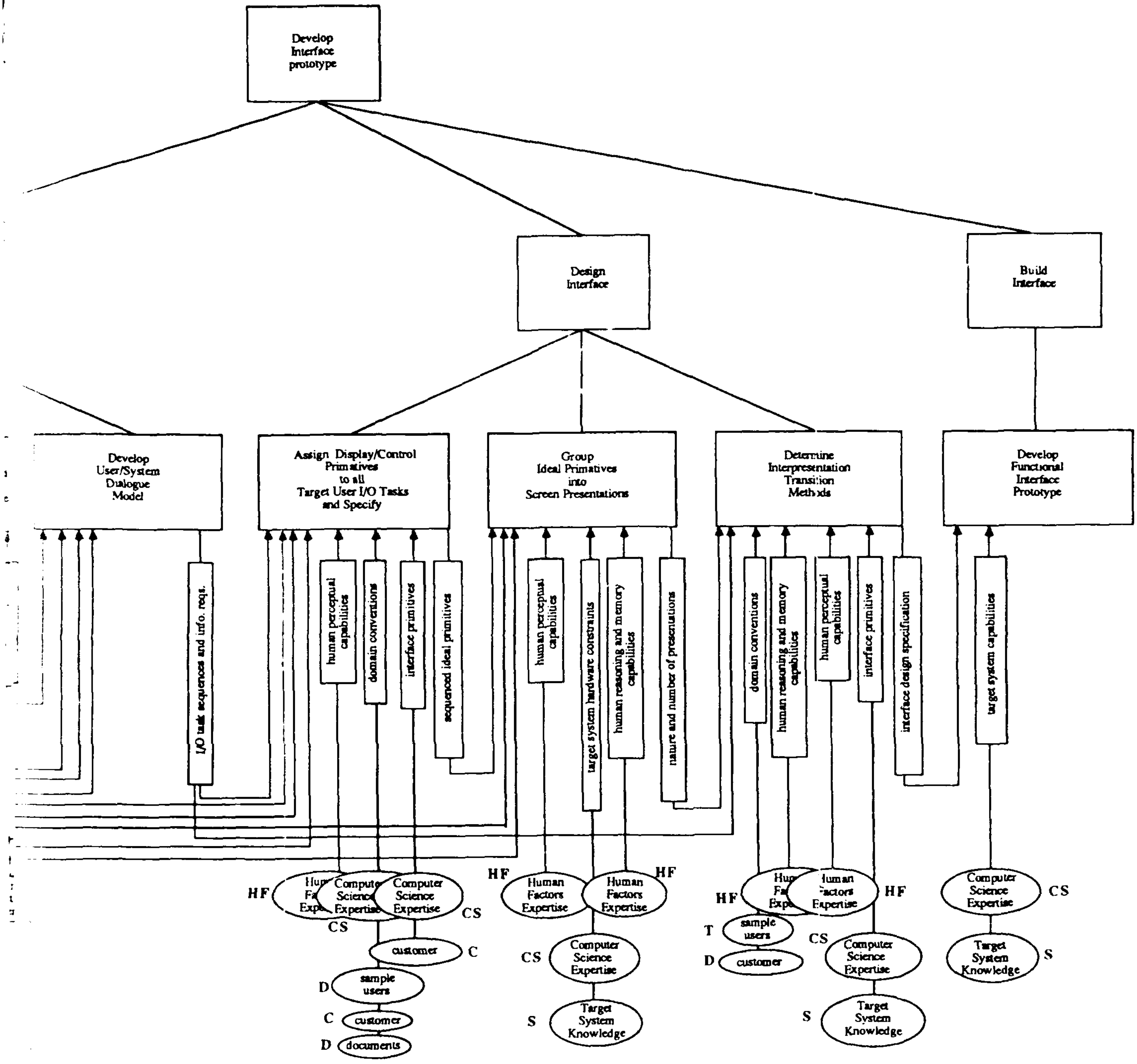

Data Source Classes
C=Customer Dictates
D=Domain Knowledge
CS=Computer Science Expertise
HF=Human Factors Expertise
S=Application System Knowledge
T=User Task Knowledge
Develop Interface prototype
Define Requirements
Construct Task Model
Allocate Functions to User or System
Develop User/System Dialogue Model
Assign Display/Control Primatives to all Target User I/O Tasks and Specify
knowledge acquisition results
domain conventions
customer dictates
goals
procedures
information and flow
contingencies
customer dictates
target user skill levels
human reasoning and memory capabilities
target system capabilities
human vs system responsibilities
I/O task sequences and info. reqs.
human perceptual capabilities
domain conventions
interface primitives
sequenced ideal primitives
HF
Human Factors Expertise
T
sample users
D
sample users
D
documents
C
customer
C
customer
customer
C
D
sample users
Computer Science Expertise
CS
C
customer
HF
Human Factors Expertise
Target System Knowledge
S
HF
Human Factors Expertise
Computer Science Expertise
CS
Computer Science Expertise
CS
HF
customer
C
D
sample users
C
customer
D
documents

Figure 3.3 Best-case interface design model.


Develop Interface prototype
Design Interface
Build Interface
Develop User/System Dialogue Model
Assign Display/Control Primatives to all Target User I/O Tasks and Specify
Group Ideal Primatives into Screen Presentations
Determine Interpresentation Transition Methods
Develop Functional Interface Prototype
I/O task sequences and info. reqs.
human perceptual capabilities
domain conventions
interface primitives
sequenced ideal primitives
human perceptual capabilities
target system hardware constraints
human reasoning and memory capabilities
nature and number of presentations
domain conventions
human reasoning and memory capabilities
human perceptual capabilities
interface primitives
interface design specification
target system capabilities
HF
Human Factors Expertise
Computer Science Expertise
Computer Science Expertise
CS
CS
customer
C
D
sample users
C
customer
D
documents
HF
Human Factors Expertise
Human Factors Expertise
HF
CS
Computer Science Expertise
S
Target System Knowledge
HF
Human Factors Expertise
Human Factors Expertise
HF
T
sample users
D
customer
CS
Computer Science Expertise
S
Target System Knowledge
Computer Science Expertise
CS
Target System Knowledge
S

representatives of the domain to review and critique, perhaps several times, the products of the analysis before moving to the next step. In situations where there is a requirement to design the system in such a way that it will change current practice, the modelling effort is more difficult, and usually requires additional iterations that include the sponsor as well as representatives of the domain. In the simplest case, knowledge of the task, the task domain, and the analysis techniques are required to construct a task model. When changes to the current practice are desired, additional information about the desired modifications is required.

3.2.1.2 Allocation of Functions

The next step requires both knowledge of the task that the target system will support and the functional capabilities of the application that the user will interact with through the interface. These two classes of information, combined with knowledge concerning the relative capabilities of the intended class of user versus the application are used to make decisions concerning the assignment of responsibilities associated with each task described by the task model developed in the previous step. This activity, like task analysis, is commonplace for engineers developing large scale interactive systems but as yet, is not often an explicit step in the development of interactive software (Kantowitz and Sorkin 1987; Meister 1985).

Like task modelling this process will usually require several iterations before it is completed, and its results may suggest requirements for the application that were not foreseen initially. The model not only requires human factors expertise, and knowledge of the task and domain, it also requires considerable judgement concerning the expected capabilities of the application (Meister 1988).

It is reasonable to expect the outcome of this process to be the same tables or graphs used to describe the task model constructed from the previous analysis with the addition of annotations for each task description classifying it either as an operator or an application function. User opinions are useful to solicit at this point. They may have unique insights into the difficulties that may be encountered when attempting to automate certain functions, or they may express preferences for leaving certain processes under the control of the user for reasons that may not be obvious to the analyst.

3.2.1.3 Dialogue Specification

The next recommended design task is to incorporate modifications expected to result from the introduction of the technology under development into the task model by defining the user I/O tasks that the interface must support. This model should include descriptions of the data characteristics, such as its scale, (for example, categorical, or interval, or ratio scale), as well as the sources of this information within the application design and the interface protocols. Many high-level, executable languages have been development to catalogue the results of this process. This cataloguing includes graphic network views, as well as text-based descriptions that can be directly incorporated into the final interface code for run time execution. If directly incorporated, the specific syntax of the language can constrain the style of interaction that would be possible, but, experience has shown that they can be very efficient for reducing interface implementation time if they are compatible with the nature of the task and style of interaction that the application requires (Green 1986). Unfortunately, most of the available languages were developed before the advent of the direct-manipulation dialogue style, and they have trouble supporting it (see Chapter 4).

In many actual development efforts, user and task modeling do not precede the definition of the dialogue. In these cases, the interfaces which result tend to present a logical structure to the user that more closely mirrors the organization of the underlying application than the task it is intended to support. This can constrain the use of the system in ways that are incompatible with the user's natural approach to problem solving. For this reason many recommend that the task model serve as the basis for defining sequencing and transition constraints.

At the time of this writing, there do not appear to be any dialogue specification languages currently available or under development which incorporate specific task analysis formalisms as well as dialogue definition syntax. At a superficial level, there does not appear to be any technical reason that precludes their development, since graphic languages designed to support task analysis, procedural knowledge simulations, and executable dialogue descriptions are commercially available.

3.2.1.4 Assignment of Interface Primitives to I/O Transactions

Once a dialogue has been specified for data transfer and system control actions, the actual appearance of the interface can be defined by specifying the interface elements that will be available to the user for viewing and entering data, and asserting commands at each of the transaction points in the dialogue model (eg. windows, forms, lists, menus, charts, graphs, tables, spreadsheets, maps, icons, labels).

Ideally these judgements should be made from knowledge of the domain conventions for data presentations and concept representation as well as expertise concerning the relative effectiveness of various methods of control and display from a human performance point of view. Domain conventions can be identified during the task modelling activity, and human engineering guidelines for these assignments are available from the various sources discussed in Section 3.1. Some principles, such as consistency of use and immediacy of feedback are universally applicable; however, the majority of guidelines are oriented to alphanumeric applications at this point, and often are difficult to interpret without substantial experience with their application. Unfortunately, research to formalize the relationship between task objectives and interface technology has lagged behind its availability and application, rendering many aspects of this design task as much an art as a science.

Ideally these methods should be determined primarily from a human performance point of view, and compromised according to other constraints as infrequently as possible. Hardware and software constraints, for example, will invariably influence the relative value of the various methods that could be used (for example, window support, pointing devices, monochrome vs color screen, and bit mapped graphics) as will the costs associated with using them. In the case of the development of the Apple Lisa interface, usability objectives were given such high priority that, in several instances, the respecification and design of the underlying hardware environment was undertaken.

3.2.1.5 Aggregation of Primitive Elements into Presentations

The organization and appearance of the interface elements associated with the dialogue transactions are specified in this step. Considerations of grouping and display clutter (Tullis 1987) in conjunction with viewport or screen size will determine the upper bound on the number of elements included in each presentation. The specific content of the presentations should be based on the sequential aspects of the task model, as well as the process "chunking" implied by the goals and subgoals associated with the problem solving tasks and the dialogue transactions required to accomplish them (Norman, Weldon, and Shneiderman 1986). User task interdependencies must to be considered to insure that the information required for each problem step is available within a single presentation (Henderson and Card 1986). In situations where dependencies occur between presentations, redundant information may need to be provided to help the user transition between presentations without losing his/her task context (Woods 1984).

3.2.1.6 Selection of Interpresentation Transition Methods

Specification of the methods that will be used to select and move between the presentations specified in the previous design step that afford the user the maximum degree of flexibility are developed next. The organization of these controls should consider the sequential aspects of the application task and at the same time be selected so as to minimize the number of steps the user has to take when moving between presentations.

Human engineering guidelines are available for selecting methods that have been shown to be effective for alphanumeric or form-filling applications (Smith and Mosier 1987), but it is often a necessary to override these recommendations if the nature of the task being supported suggests a structure that violates these principles (Bury 1982). Navigation aids that depict the options available at each of the various presentations should be considered (Card and Henderson 1987), as well as graphs which specifically depict the overall structure of the system and interface (Billingsley 1981). These include, for example, world-view maps of the interface organization or utilities that enable the user to step from presentation to presentation in a sequence that is typical of normal task accomplishment. Menus systems that are broad rather than layered are generally preferred, as are item names that are common in the task domain.

3.2.1.7 Prototype Development

The final step in the first cycle of design involves the construction of a prototype interface that will allow users to experiment with some or all aspects of the interface function, including its "look and feel." At the lowest level of prototype fidelity is, the graphic "storyboard" approach, which depicts only the look of the presentations but has been shown to be useful for eliciting user impressions. Interactive presentations that simulate both the look and feel of the interface offer the next level of realism, followed by facades which incorporate simulated system responses and response times. The highest fidelity and most useful prototype will include the running application software and additional data collection routines for quantitative user-system performance evaluations.

Each of the prototyping methods described has strengths and weaknesses that make it difficult to rank them. The storyboard approach is the least costly in terms of the time and skill required. Facades take more time to produce, but for evaluation purposes provide the dimension of feel that storyboards do not. Finally, the fully functional prototype provides the most realism but at the highest cost in terms of time, programming, and skill requirements. In many instances, the uncertainty associated with initial designs do not warrant the investment in completely operational prototypes.

3.2.2 Worst-Case Model

For most interface designers, the worst-case model is probably more typical of what the actual interface development process is like today. Here we assume an emphasis on usability later in the system development process, when time and resource constraints are critical. We assume that the programmer's interface used to support the development of the application is in place before user-oriented design questions are seriously considered, and that a substantial proportion of the development resources have already been consumed before these questions are addressed. This process involves more versions of functional software early in the development process than the best-case model, and less end-user involvement.

The description of the worst-case development process is broken into stages of establishing changes to the programmers' interface, determining the resources that can be applied, locating software that can be used with a minimum of modification, locating software that can be adapted, prioritizing the remaining shortcomings, and developing some specific new interface tools with the resources that remain (see Figure 3.4). Like the best-case model, the worst-case model is abstract and inadequate for use as the knowledge-base for an interface design tool. It is offered as an alternative hypothetical description of what the outcome of a knowledge engineering effort of this type might look like, in terms of the types of knowledge, activities, and sequences of actions that would be used, and conforms more closely to a process of innovative design than to invention.

3.2.2.1 Establish Required Changes

The primary objectives that most software development projects focus on, at least in the earliest stages, is getting the system to function in a way that satisfies the customer. In order to do this the developers often construct an interface to the application that is being built, structured not in terms of the user's point of view, but in terms of the underlying software, with a focus on facilitating program tests and implementation. In some instances, the interfaces are developed using a rapid prototyping environment that provides exceptionally high bandwidth interface tools. These tools enable the programmers to view the development from multiple perspectives depending on the development need. This flexibility is necessary from the programmer's point of view and is likely to offer more control and display options than an end user would need. But this same flexibility can make the system appear virtually unstructured and very difficult for the casual user to operate.

Initial customer demonstrations are often when it becomes apparent that this high bandwidth interface is difficult for people to use. Fears that the customer will not be able to understand what the application does, let alone use it, develop along with the first serious consideration of using human engineering principles. The problem with this approach to usability is that the prior emphasis on system function did not allow sufficient time and effort to be applied to the problem of understanding in detail how users of the system would prefer to have the interface structured, leaving the human engineering activity to be based more on prior experience and intuition than on systematic analysis of the users' task and the conventions from the domain that could be exploited.

Nevertheless, the identification of usability problems are typically followed by general human engineering evaluations carried out by engineers who have had minimal exposure to users' or the domain. A typical result of such an evaluation is a list of changes that will improve the interface from the users' point of view. Programmers tend to resist these recommendations when they involve reworking the software that has already been produced. Some subset of this list of changes is usually accepted as worthwhile by the project team, and plans to implement them are developed.

3.2.2.2 Determine Available Resources

The need for changes often surfaces after substantial resources have been committed to the development of the application software. This usually takes place at critical times just before customer evaluations. The upcoming customer demonstration tends to motivate the development team to develop those interface enhancements that can be used by programmers to present the power of the application program rather than to develop the interface that will support its use by nondevelopers. Therefore, the budget that appears to be allocated to usability issues, is, in fact, many times being channeled to demonstration objectives that often turnout to be less useful to end users than expected. This is because programmers are still the primary operators of the system.

3.2.2.3 Identify Existing Interface Software That Can Be Applied

Once time and budget issues are worked out, the fastest and least costly way to overcome the complex programmers' interface is to leverage software that has been developed for another purpose and apply it to the current problem. In fact, this "innovative" style of design is so common, that the capability of a piece of software to be reused is often a major criterion used to assess software quality.

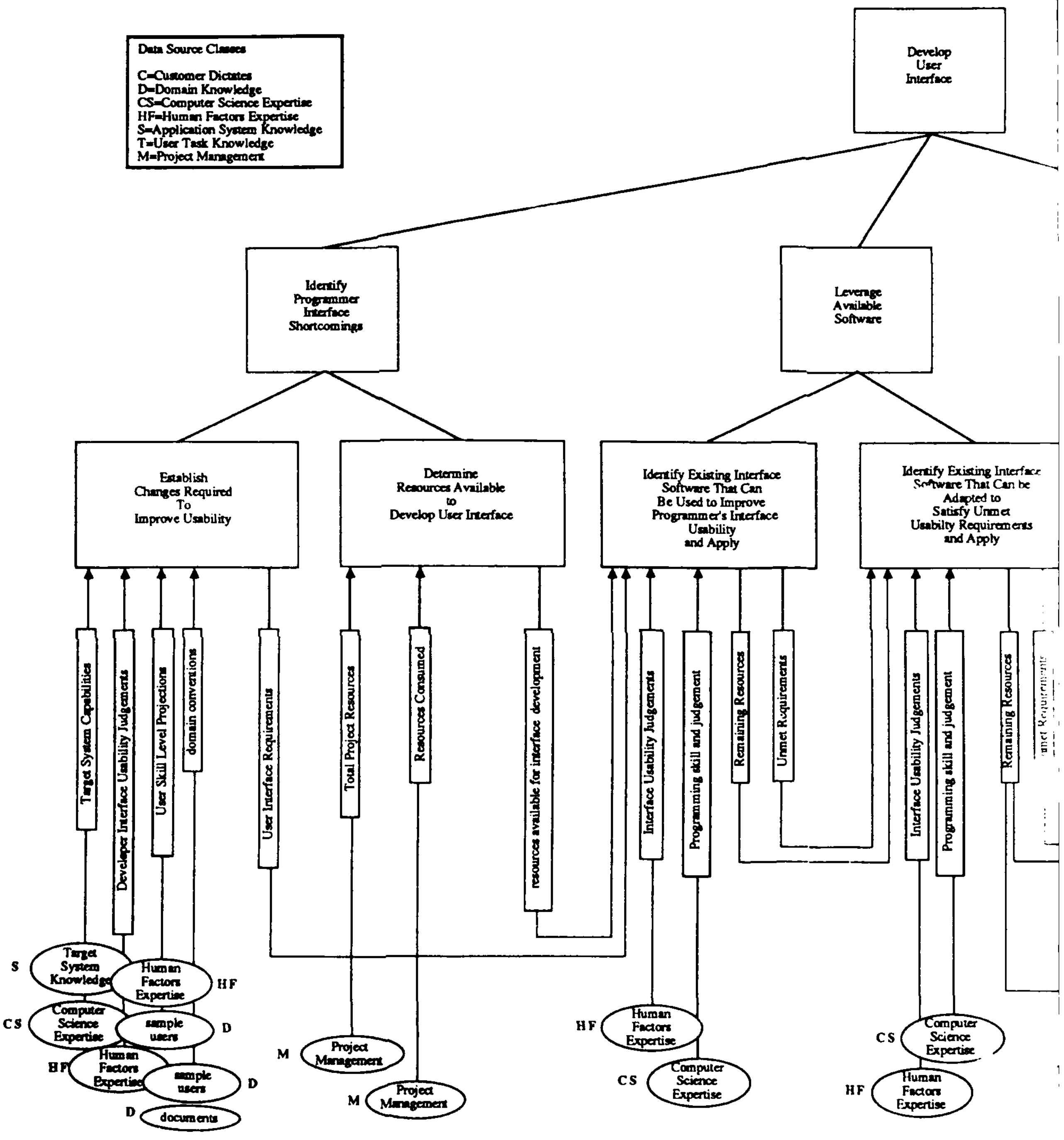
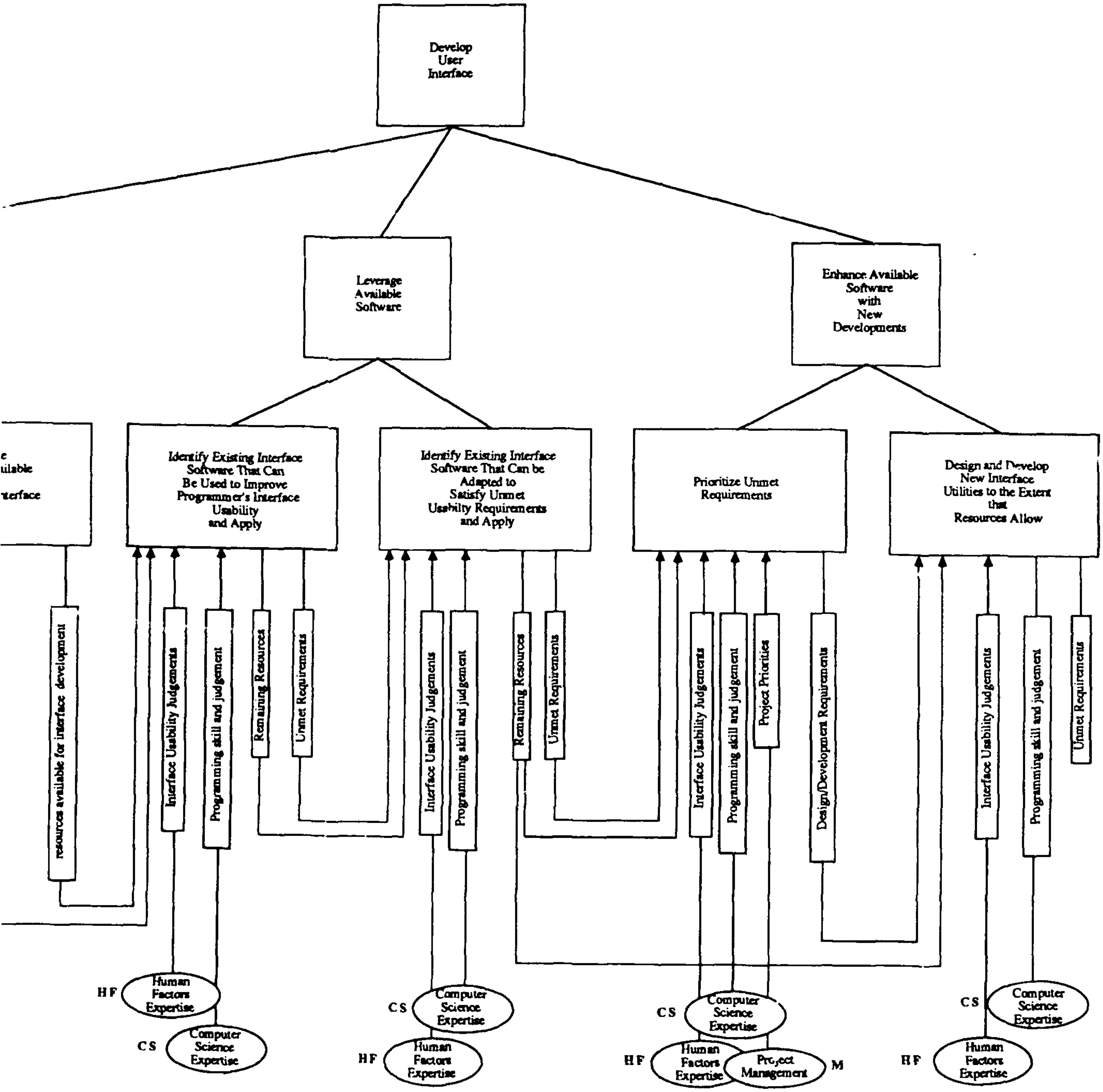

Data Source Classes
C=Customer Dictates
D=Domain Knowledge
CS=Computer Science Expertise
HF=Human Factors Expertise
S=Application System Knowledge
T=User Task Knowledge
M=Project Management
Develop User Interface
Identify Programmer Interface Shortcomings
Leverage Available Software
Establish Changes Required To Improve Usability
Determine Resources Available to Develop User Interface
Identify Existing Interface Software That Can Be Used to Improve Programmer's Interface Usability and Apply
Identify Existing Interface Software That Can be Adapted to Satisfy Unmet Usabilty Requirements and Apply
Target System Capabilities
Developer Interface Usability Judgements
User Skill Level Projections
domain conventions
User Interface Requirements
Total Project Resources
Resources Consumed
resources available for interface development
Interface Usability Judgements
Programming skill and judgement
Remaining Resources
Unmet Requirements
Interface Usability Judgements
Programming skill and judgement
Remaining Resources
nmet Requirements
S
Target System Knowledge
Human Factors Expertise
HF
CS
Computer Science Expertise
sample users
D
HF
Human Factors Expertise
sample users
D
D
documents
M
Project Management
M
Project Management
HF
Human Factors Expertise
CS
Computer Science Expertise
CS
Computer Science Expertise
HF
Human Factors Expertise

Figure 3.4 Worst-case interface design model.


Develop User Interface
Leverage Available Software
Enhance Available Software with New Developments
e
ilable
terface
Identify Existing Interface Software That Can Be Used to Improve Programmer's Interface Usability and Apply
Identify Existing Interface Software That Can be Adapted to Satisfy Unmet Usability Requirements and Apply
Prioritize Unmet Requirements
Design and Develop New Interface Utilities to the Extent that Resources Allow
resources available for interface development
Interface Usability Judgements
Programming skill and judgement
Remaining Resources
Unmet Requirements
Interface Usability Judgements
Programming skill and judgement
Remaining Resources
Unmet Requirements
Interface Usability Judgements
Programming skill and judgement
Project Priorities
Design/Development Requirements
Interface Usability Judgements
Programming skill and judgement
Unmet Requirements
HF
Human Factors Expertise
CS
Computer Science Expertise
CS
Computer Science Expertise
HF
Human Factors Expertise
CS
Computer Science Expertise
HF
Human Factors Expertise
Project Management
M
CS
Computer Science Expertise
HF
Human Factors Expertise

Human engineering principles do not play a major role in this process beyond the initial list of requested enhancements made previously. The focus of the effort is primarily on improving the appearance of the system interface using the minimum amount of resources possible, with human factors constraints taking a secondary role to resource conservation. Decisions at this point primarily involve computer programming judgement concerning the compatibility of software interfaces between the new and old software, rather than the long term suitability of the interface. At this point, most team members, including the human engineering people agree that it is more important that the customer and potential users get a timely look at the system than it is to perfect the user interface software.

3.2.2.4 Identify Existing Interface Software that Can be Adapted

After reusing existing software, the most expedient and least expensive way to enhance the appearance of an application further is to adapt interface software developed for other purposes that may not initially be directly compatible with the current application. Again, the priority here is demonstration rather than use, which makes these short term solutions appear superior to the longer term, more appropriate solutions that would require more time and resources to implement.

Beside the fact that judgements are made without incorporating much attention to realistic user skill and compatibility requirements, the major shortcoming of this practice is that the process of adaptation invariably takes more time and effort than is allocated for it. Sometimes it can cost even more than the development of completely new software. This approach leaves fewer resources for a user-oriented solution, making it likely that the expedient innovation will have to remain a part of the final product.

3.2.2.5 Prioritize Remaining Unmet Requirements

The costs associated with developing interface utilities "from scratch" are usually at least as high as planned. Whenever possible, decisions to invest in them are based on both of the immediate project's needs that they will serve and on needs projected for systems that the organization might build in the future. Because there is an interest in making these new interface utilities reusable for future applications, they are often designed to be more general and less "tuned" to the specific needs of the current application and they are often more time consuming, in the short run, to produce. Because of these factors, the usability of these new interface utilities can be compromised.

3.2.2.6 Design and Develop New Interface Utilities

There is usually an interest in ensuring that the interface enhancements that are made have as many human engineering principles built into them as possible. Unfortunately, the objectives of making these interfaces as general as possible can override requests to tailor the interfaces to the needs of the immediate application. This fact, combined with limited knowledge of the particular application task that it will be applied to in the future, make it difficult for human engineering principles, task characteristics and domain conventions to play a major role in their design.

The limited resources applied to the user interface, the late phase of development when human engineering principles are finally applied to the problem, and the pressures to demonstrate functionality as early in the project as possible each can make it difficult to produce high quality user interfaces. The outcome of a development effort in which all of these problems are manifested is usually a system that appears to have lots of potential that never gets into the hands of the user community in a form that allows this potential to be fully realized. However, until adequate resources are appropriated, and until sponsors are willing to wait longer before demanding prototype previews, the worst-case "innovative" development model, we have described in this Section, is likely to predominate over the best-case "invention" model presented previously.

3.3 Conclusions

Several points have been made in this chapter concerning interface design guidelines and the process of interface design that are relevant to specifying requirements for a knowledge-based interface design tool

First of all, we have shown that the voluminous and diffuse literature associated with human factors engineering and user interfaces has a history of problems associated with its application by both specialists and non-specialist to design problems. We have identified two attempts to use human engineering guidelines as a basis for knowledge-based design tools that demonstrate that this literature is in a form that requires further knowledge engineering before it is suitable for this application.

Guidelines have been shown to be difficult to apply without considerable human factors background knowledge and to call for and require supplementary information about the specific application, task, and domain being supported. One attempt to develop a rule-based system for interface design using the available guidelines raised questions concerning the usefulness of the system because of the degree to which the rules required context-defining data to be entered by the user before its recommendations could be made. Finally, some of the most important emerging areas of interface design, such as the appropriate use of graphic display and direct manipulation techniques, are just beginning to be studied. It may be some time before this research is completed and the results are in a form that is directly applicable to the design process.

A second set of conclusions surrounds the process by which interface design guidelines are used to influence software design. Design in general is a complex form of problem solving that is not well understood outside of specific, well-defined problem areas, and there appears to be a wide range of ways that a given design problem can be solved. The real-world user-interface design processes appears to contrast sharply with the somewhat narrow, academic, and idealistic prescriptive models existing in the literature. These models tend to exclude issues surrounding the software, organizational, and cost factors that real world designers must contend with, in addition to issues of dialogue design, when working as part of a multidisciplinary team of system developers that is concerned with the functionality of the applications as well as its usability.

This deficiency of the literature led us to develop two models of the process that recognize the influence of more factors than most prescriptive models of interface design and emphasize different styles of development. Our own experience leads us to advocate the best-case model, but cultural and organizational factors are likely to cause the process to look more like the worst-case model in most real world situations. As a consequence, it is our opinion that interface development tools should ideally be able to support the best-case process without necessarily requiring it to be followed in total or in any particular sequence, and to also provide support for the innovative design model.

One could attempt to enforce a nominal design process, like the best-case model, through the design of the tool, as was attempted with the KADD system (Frey and Weiderholt 1986). However, designer evaluations of KADD indicated that the system would be have been more acceptable and useful to them if it was designed to support a design process closer to the worst-case model. They reported that if it constrained them in ways that would require them to expend more time, or to expend more effort than they normally do during the design process, that it probably would not be used in the field, even if it produced better products (Hunt and Frey 1987).

4. Interface Design Support Tools-State of the Art

In this chapter we survey state-of-the-art user interface design software tools through a literature review devoted to user interface management systems (UIMS), adaptive interfaces, and intelligent interfaces. We survey the UIMS literature because design support is an explicit part of their function (Betts, Burlingame, Fischer, Foley, Green, Kasik, Kerr, and Olsen 1987), and review adaptive and intelligent interface literature because many of these systems have knowledge-based architectures that incorporate explicit models of tasks, users, domains and applications, for selecting and formatting presentations.

The discussion of the state of the art is organized in terms of the design tasks identified in the best-case interface design process model presented in Chapter 3, which include task model construction, function allocation analysis, dialogue modeling, development and assignment of I/O interface primitives, organization of primitives into screen presentations, definition of presentation selection methods, and prototype construction. In Tables 4.1a and 4.1b we present descriptions of 20 systems in terms of the extent to which they support these design tasks. Each of these systems is reviewed in more detail in Appendix B. A list of commercially available interface design products is provided in Appendix C.

4.1 Task Model Construction

Tools for developing task models fall into two categories, methods of analysis and schemes for the representation of the results (Meister 1985). As discussed in the previous chapter, various task analysis methods have been used by human factors engineers to support the design of interactive systems for many years and the recognition of the relevance of these techniques to supporting the conceptual design and requirements definition phases of software interface development projects is growing (Bennett, 1987; Rhyne, Ehrich, Bennet, Hewett, Sibert and Bleser, 1987).

Problems associated with the development of direct-manipulation interfaces in particular are making the need for user, domain, and task models more apparent, with some suggesting that their construction should play a role in the interface design process as central as knowledge engineering is to the design of expert systems (Betts, Burlingame, Fischer, Foley, Green, Kasik, Kerr, Olsen, and Thomas 1987). Interface development environments that support the definition of interface requirements through task models that can be interpreted to automaticaliy generate interface designs have also been suggested as a way to make the next generation of user interface management systems compensate for deficiencies in the level of abstraction at

Interface Design Functions / Systems	Construct a Task Model Perform decomposition of the user problem that the system is being designed to facilitate. Determine user goals, tasks that will enable the accomplishment of these goals, the information regulred to perform these tasks, the information produced as a consequence of these tasks, situation specific process dependencies, and sequential processing dependencies	Allocate Functions to User or System Using the task model, identify those aspects of the problem that the system will handle and those aspects that the user will be responsible for This should include information provision and storage, task execution, and goal/state evaluations This requires knowledge of the projected or known functional capabilities of the underlying application system	Develop User/System Dialogue Model Based on the task model and the allocation of function analysis, produce a description of the problem solving process as it will be accomplished using the system being designed, to define the user's I/O tasks that the interface must support	Assign Display/Control Primitive to all Target User I/O tasks and Specify Determine the appropriate and/or feasible types of display-control primitives to support each I/O activity defined in the dialogue model This requires application human factors expertise to dictate optim approachs or the selection of satisfactor approaches from within the constrained s of available interface utilities
Arens et al. 1988 Intergrated Interfaces			Supports models that contain descriptions of the type of information the application progam deals with and of the type of graphical tools and instruments available	Applies rules derived from application models and interface models to constuc a presentation interface
Reach and Stone 1983 TiegaArtwork				
Borning and Duisberg 1986				
Corbett 1985 CHIPS				
Ferrante and Tench 1986 TIDAS				Provides a display editor to enable the user to manipulate the definitions of overall window size, aspect ratios and the position size and type of any con pane
Flanagan et al. 1987 RIPL			Requires a detailed user specification of the user-system dialogue in terms of a set of explicitly linked state/screens with associated information lists. This dialogue model is independent of the particular I/O utilities prototyped for each interaction specified	Provides a graphics editor to enable the user to manually draw items on tiles and a utility to import graphics from Macintosh paint programs An on-line version of the Smith and Mos guidelines document is provided
Foley et al. 1988 UIDE				Supports libraries of a range of functional graphic interface objects and prototype designs
Frey and Wiederholt 1986 KADD	Provides a language to support the decomposition of the users task as a way to input interface information requirements			Provides a library of high-level graph display object primitives for aircraft cockpit applications and a limited human factors knowledge base used to evaluate their use
Friedell 1984				Supports automatic icon generation th the use of applying synthesis operators object descriptions
Gargan et al. 1988			Provides a dialogue model in which the user interacts and the system extracts knowledge of the presentation	Applies rules derived from user models select modalities

Table 4.1a Summary of informal review of 20 interface design

Develop User/System Dialogue Model Based on the task model and the allocation of function analysis, produce a description of the problem solving process as it will be accomplished using the system being designed, to define the user's I/O tasks that the interface must support	**Assign Display/Control Primitives to all Target User I/O tasks and Specify** Determine the appropriate and/or feasible types of display/control primitives to support each I/O activity defined in the dialogue model. This requires application of human factors expertise to dictate optimal approachs, or the selection of satisfactory approaches from within the constrained set of available interface utilities	**Group Ideal Primitives Into Screen Size Presentations** Apply display real estate and human factors constraints to the selected display control methods. Format high level presentations (screens or windows) according to user goals, data flow and sequential contraints using as few presentations as possible	**Determine Interpresentation Transition Methods** Define methods for presentation selections (menus, commands etc). Apply human factors expertise to make the identification and selection of presentations easy, the transitions smooth and intuitive, and the information presented appear contiguous	**Develop a Functional Interface Prototype** Construct a prototype interface that allows users to experiment with the function, look, and feel of it. Incorporate the performance characteristics of the underlying system to provide valid response times etc. and provide data collection routines to support user-system performance evaluations
upports models that contain escriptions of the type of information e application progam deals with and of e type of graphical tools and ments available	Applies rules derived from application models and interface models to constuct a presentation interface			
		Rules to format graphics with a consistant style (designed for document graphics)		
		Maintains consistancy of the interface by using the bidirectionality of constraint relations		Supports the development of broad utility prototype interfaces using object libraries and at a lower level supports the creation of such libraries
				Supports the develpment of broad utility prototype interfaces using object libraries and at a lower level supports the creation of such libraries
	Provides a display editor to enable the user to manipulate the definitions of overall window size, aspect ratios, and for the position size and type of any component pane	Maintains consistancy of the interface by using the bidirectionality of constraint relations	Provides functional presentation control objects through the use of a simulator so the user can ge a "look and feel" of how the display will preform	Supports the design of fully functional user interface prototypes that can be applied to operation systems
es a detailed user specification e user system dialogue in terms of et of explicitly linked state/screens associated information lists s dialogue model is independent of the I/O utilities prototyped teraction specified	Provides a graphics editor to enable the user to manually draw items on tiles and a utility to import graphics from Macintosh paint programs An on-line version of the Smith and Mosier guidelines document is provided	Provides a metric evaluator to check user defined tile attributes for consistency in terms of positional consistency, stimulus consistency and screen density	Provides a 240 rule MYCIN like expert system that the user interacts with to manually define a dialogue problem. System has knowledge of and can recommend either a menu, command, or icon based transition method	A facade's tiles interaction dynamics can be simulated with specified system response times. Data from user interaction sessions can be collected and behavior logged for session play-back
	Supports libraries of a range of functional graphic interface objects and prototype designs	Presentation designs are defined by a series of transformations	Provides functional presentation control objects and actions through the use of semantic routines	Supports the design of fully functional user interface prototypes that can be applied to operational systems
	Provides a library of high-level graphic display object primitives for aircraft cockpit applications and a limited human factors knowledge-base used to evaluate their use			Supports the mocking-up of non-functional displays that can then be assessed in terms of their "look."
	Supports automatic icon generation through the use of applying synthesis operators to object descriptions			
es a dialogue model in which the interacts and the system extracts wledge of the presentation	Applies rules derived from user models to select modalities		Applies rules derived from user models to select techniques to present information within a selected modality	

ry of informal review of 20 interface design support systems.

Interface Design Functions / Systems	Construct a Task Model Perform decomposition of the user problem that the system is being designed to facilitate. Determine user goals, tasks that will enable the accomplishment of these goals, the information regured to perform these tasks, the information produced as a consequence of these tasks, situation specific process dependencies, and sequential processing dependencies	Allocate Functions to User or System Using the task model, identify those aspects of the problem that the system will handle and those aspects that the user will be responsible for. This should include information provision and storage, task execution, and goal/state evaluations. This requires knowledge of the projected or known functional capabilities of the underlying application system.	Develop User/System Dialogue Model Based on the task model and the allocation of function analysis, produce a description of the problem solving process as it will be accomplished using the system being designed, to define the user's I/O tasks that the interface must support	Assign Display/Control Primitives to all Target User I/O tasks and Specify Determine the appropriate and/or feasible types of display/control primitives to support each I/O activity defined in the dialogue model. This requires application of human factors expertise to dictate optimal approachs, or the selection of satisfactory approaches from within the constrained set of available interface utilities	Gr Sc App con met (scr goa usin
Hayes 1985 COUSIN/SPICE			An event-based dialogue is implicitly specified when a form filling style template is instantiated for a particular application using a high level text-based language		
Jacob 1985			Provides a comprehensive, executable high-level user/system dialogue specification language with a graphic visual component		The inter spe
Lockheed 1987 LUIS				Supports libraries of a range of functional graphic interface objects without assistance in their appropriate application	
MacKinlay 1986 APT				Applies some rules derived from graphic design principles of Bertin and theories of Cleveland and McGill to select and format graphs	
TULLIS 1986					
Tyler 1988 SAUCI				Applies rules derived from knowledge bases to determine the values of certain parameters for adapting the interface	
Virtual Prototypes Inc. 1987 VAPS					
Wasserman and Shewmake 1985 RAPID/USE			Supports a text and graphics dialogue description language based on extended state transition diagram models to support strictly alpha numeric interfaces	Provides a moderately high-level language for specifing alpha-numeric displays, the requirements for which are defined by the state transition diagrams	Pres def
Weitzman 1986 DESIGNER				Provides a library of high-level graphic interface primitives developed for the STEAMER domain	Allow have of th asso impro incon
Yunten and Hartson 1985 SUPERMAN			A prototype graphic specification language and editor that can depict function, data flow, control flow, and system vs user functions in a single representation scheme at multiple levels of abstraction		

Table 4.1b Summary of informal review of 20 interface design suppo

will will This	**Develop User/System Dialogue Model** Based on the task model and the allocation of function analysis, produce a description of the problem solving process as it will be accomplished using the system being designed, to define the user's I/O tasks that the interface must support.	**Assign Display/Control Primitives to all Target User I/O tasks and Specify** Determine the appropriate and/or feasible types of display/control primitives to support each I/O activity defined in the dialogue model. This requires application of human factors expertise to dictate optimal approachs, or the selection of satisfactory approaches from within the constrained set of available interface utilities	**Group Ideal Primitives into Screen Size Presentations** Apply display real estate and human factors constraints to the selected display control methods. Format high level presentations (screens or windows) according to user goals, data flow and sequential contraints using as few presentations as possible	**Determine Interpresentation Transition Methods** Define methods for presentation selections (menus, commands etc). Apply human factors expertise to make the identification and selection of presentations easy, the transitions smooth and intuitive, and the information presented appear contiguous.	**Develop a Functional Interface Prototype** Construct a prototype interface that allows users to experiment with the function, look, and feel of it. Incorporate the performance characteristics of the underlying system to provide valid response times etc. and provide data collection routines to support user-system performance evaluations
	An event-based dialogue is implicitly specified when a form filling style template is instantiated for a particular application using a high level, text-based language				Supports the development of alpha-numeric prototype interfaces through a high-level text-based language used to manipulate form-based primatives and I/O utilities such as pop-up menus buttons and field definitions. For some applications, the code can serve as the final user interface
	Provides a comprehensive, executable high-level user system dialogue specification language with a graphic visual component		The sequential aspects of the interface dialogue are defined through the specification language		
		Supports libraries of a range of functional graphic interface objects without assistance in their appropriate application		Provides functional presentation control objects such as menus and pop-up windows	Supports the design of fully functional user interface prototypes that can be ported and applied to operational systems Performance data collection is supported and can be organized in terms of a goal-oriented dialogue model
		Applies some rules derived from graphic design principles of Bertin and theories of Cleveland and McGill to select and format graphs			
				Evaluates p e-existing alpha-numeric displays in terms of overall density, local density number of visual groups, average visual angle subtended by groups, and layout complexity to predict search time and subjective opinions using regression models Provides diagnostic displays and recommends changes based on deviations from norms	
		Applies rules derived from knowledge bases to determine the values of certain parameters for adapting the interface			Can be used as a separate textual front-end interface for an operational system Provides an int-lligent advising facility to the user for direction in use of application commands
					Provides menu and window access to a graphics editor, logic editor, integration editor and runtime environment for the creation of displays and controls that can access simulation models or operational systems Applications have included cockpit and ship controls, C3I and ATC
	upports a text and graphics dialogue description language based on extended state transition diagram models to support strictly alpha-numeric nterfaces	Provides a moderately high-level language for specifing alpha-numeric displays, the requirements for which are defined by the state transition diagrams	Presentations are defined by state definitions	Presentation transitions are supported by command menus	"Transition Diagram Interpreter" and "Action Linker" enable the simulation of the function, look, and feel of the interface and support extensive data collection associated with usage times keystokes, typo's etc
		Provides a library of high-level graphic interface primitives developed for the STEAMER domain	Allows the user to choose a "style" and have a STEAMER "view" critiqued in terms of the consistency of attribute values associated with it Suggestions for improvements are available when inconsistencies are found		Supports the development of functional "views" of the STEAMER mathematical model of a steam generation plant
	A prototype graphic specification language and editor that can depict function data flow, control flow, and system vs user functions in a single representation scheme at multiple levels of abstraction				A compiler for the graphical specification language produces Fortran code, thereby supporting aspects of the interface development process following specification

mmary of informal review of 20 interface design support systems (continued).

which their users must define their designs, and to improve their coverage of the requirement definition phase of development (Betts, Burlingame, Fischer, Foley, Green, Kasik, Kerr, Olsen, and Thomas 1987).

There is much literature devoted to knowledge engineering methods and tools in general (e.g. Boose and Gaines 1987), a smaller amount of literature discusses methods and tools to support user task modeling (Corker, Davis, Papazian and Pew 1986; Drury, Paramore, Van Cott, Grey, and Corlett 1987; Laughery, and Laughery 1987; Mancini, Woods, and Hollenagel, 1987; Pew, Baron, Feehrer and Miller 1977; Rouse, 1980), and still less literature relating task analysis to user interface design (Norman, 1986; Papazian 1986; Rubinstein and Hersh 1984; Williges, Williges, and Elkerton, 1987).

In many respects, a user-system dialogue model implicitly incorporates aspects of a user task model, but there are critical aspects of task models that bear on display design that they typically ignore. For example, the functions or goals that the user seeks to achieve through the use of the data, procedures, and devices will affect the relative effectiveness of graphic display methods that are independent of the characteristics of the data being portrayed (Kosslyn 1985; Cleveland and McGill 1984; Simkin and Hastie 1987).

Only one of the interface development environments surveyed, the knowledge aided display design system (KADD), provides data structures for explicitly incorporating user task knowledge to support the design of displays (Frey and Wiederholt 1986). KADD's design domain is aircraft cockpit displays. It assumes a top-down, requirements driven design process, with the requirements and their interrelationships structured through a function and task analysis data base that provides for the representation of hierarchical relations among goals, tasks, and information requirements. The top level in the data base consists of the goals or functions that the end user of the interface must preform. The recorded characteristics of the goals are: a name and description, the crew members responsible for the function, the mission phases during which the function is performed, the system with which the function is performed, and the relative priority of the goal.

The second level contains records of the tasks, defined by names and descriptions, that the end user preforms to achieve the goals defined in the first level. Frey and Wiederholt (1986) report that most of the information related to the tasks comes not from the task record itself, but from what it is linked to. A task record may be linked to a goal with a relative priority and temporal relationship, to other tasks linked to the same goal, to initiating and terminating cues, or to its information requirements.

Information requirements associated with the tasks in the second level compose the third level of the task analysis data base. They are described in terms of: dimensions (for example, displacement, temperature, pressure, and speed), variables (for example, exhaust gas temperature, air speed), samples (for example, number or frequency required), uses (quantitative or qualitative), and significant ranges.

While the full task model data base can be accessed by the designer/user during the design process, only the task-specific information requirements are used by the system to evaluate and propose display designs that will satisfy them in terms of the definition of one or more basic display primitives selected from the system's library of seven high-level display elements.

4.2 Allocation of Functions

None of the systems that we have reviewed provide explicit support for the allocation of functions between human and machine. Many general tables have been published that attempt to define tasks that humans perform more effectively than machines. These tables are useful for first approximations of function allocation but more specialized tables need to be produced (Kantowitz and Sorkin 1987). However, attempts to exploit this type of information, or to provide it to designers in a form that would support the interface design process, is not reflected in any of the systems we have reviewed.

Many network dialogue models provide structures for describing the results of this process. General software requirements definition languages are available as well (Davis 1988; Yadav, Bravoco, Chatfield, and Rajkumar, 1988; Laugherty and Laughery 1987). However, the dialogue languages are, at best, appropriate for supporting the design of user interfaces that exploit conversational dialogue styles and are generally believed to provide inadequate support for iconographic, direct-manipulation interfaces.

4.3 Dialogue Specification

Since 1982, a group of computer scientists has been working to develop interface support tools that will allow designers to specify the syntactic component of user interfaces in a high-level language separate from the application software. These are called user interface management systems (UIMS).

The basic architecture, initially believed to be adequate to support these systems is presented in Figure 4.1. This model, called the Seeheim model, shows the logical components that a UIMS was believed to require when UIMS research was initiated, but now is being judged inadequate to support direct-manipulation interfaces. As will be discussed, currently the most active area of UIMS research involves the search for new architectures that will support the design of direct -manipulation interfaces. However, to date the majority of the systems developed and available for review are still based on the Seeheim model.

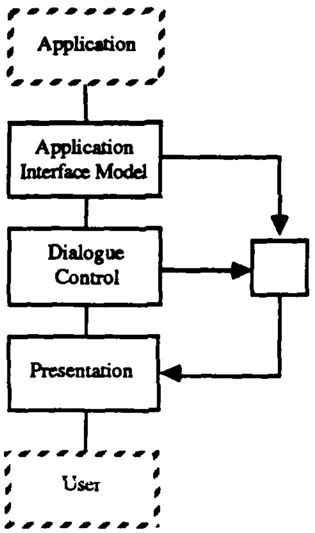


Figure 4.1 Seeheim model of UIMS (adapted from Lowgren 1988).

The presentation component of the model is responsible for the external presentation of the user interface. The dialogue control component defines the syntactic structure of the dialogue between the user and the application program, and the application interface model is used to represent the semantics of the application from the viewpoint of the user interface.

While there are still many problems associated with attempting to implement this model, the long term objective of UIMS research is to allow the generation of high quality interfaces with a minimum of effort by having the UIMS translate a designer's specification, made in terms of a specialized dialogue programming language, into a working interface (Lowgren 1988). These systems typically do not support the requirements definition phases of the design process. This is because their advocates believe UIMSs will eventually make the implementation of functional interfaces so easy that it will be more efficient to generate and test examples of working prototypes than to attempt the arduous task of defining requirements for them a priori.

In attempting to achieve this goal, many high-level dialogue specification languages have been developed in the form of transition networks, formal grammars, and event-based representations (Green 1986). Dialogue languages based on state-transition network models are the oldest, with nodes corresponding to states in the dialogue and arcs define interactions. They are functionally equivalent to grammars, but reported to be easier to read and understand, at least for relatively simple applications. For complex dialogues, with large numbers of states, hierarchical specifications with nested subnetworks have been developed (Jacob 1985, Hayes 1985). RAPID/USE takes an augmented state-transition network approach to dialogue specification (Wasserman and Shewmake 1985) also, and provides the designer with a sophisticated icon-based visual programming language for accessing the language which is intended to support the specification of form- and text- based interfaces. When languages that used the network and grammar models were developed, the dominant interface technologies being supported were "highly-moded" teletype and form-filling dialogues with strong linear components that made their structures tractable enough to make network specifications reasonable. But they are not as well suited for describing the new, relatively modeless, direct - manipulation interfaces that exploit object-oriented programming paradigms. Attempts have been made to adapt network models to these new demands (Jacob 1986), however; many have spurned this approach in favor of event-based dialogue models (Lowgren 1988).

The basic event model views the user interface as a collection of events and event handlers. An event is generated each time the user interacts with an input device. These events are processed by the event handlers associated with the input or display device involved in the interaction. The collection of events can be viewed as a state. The set of event handlers active at any one time defines the permissible user actions at that point in the dialogue. The major difference between an event-based model and a state-space dialogue specification is the degree to which the ordering between states is made explicit. In the event model, the permissible branches from one set of event handlers (state) to other sets (states) are implicit and determined at run time by the UIMS, whereas they are explicitly mapped out by the designer using a system based on network dialogue languages. Systems that use the event-based paradigm are also able to provide user feedback as a consequence of an input more efficiently. This is an essential requirement for direct-manipulation interfaces (Hutchins, Hollan. and Norman, 1986) in addition to the ability to handle concurrent events (Lowgren 1988).

Examples of UIMS systems that exploit implicit event based dialogue languages are COUSIN (Hayes 1985), IMAGES (Simoes and Marques 1987), and the University of Alberta User Interface Management System (Green 1985). And, although not UIMSs in the strictest sense of the term, the Apple Toolbox in general (Apple Computer 1985-1987) and the Hypercard/Hypertalk programming language in particular (Goodman 1987; Shafer 1988) fall within the event-based dialogue paradigm.

Other dialogue models have been proposed to support the interface design process by simply describing it using a non-executable formal grammar (for example Moran 1981). Some incorporate human performance models and explicit depictions of the problem-solving processes associated with using an interface to enable usability predictions to be derived from a design before its implementation (e.g. Card, Moran and Newell 1983). Models of this type are designed to be task-specific and also are non-executable.

4.4 Assignment of Interface Primitives to I/O Transactions

Many user interface management systems (UIMS) provide libraries and editors for the unaided use and reuse of interface primitives such as filled and unfilled rectangles, text fields, arcs, curves, bar charts, pie charts, dials, linear scales, buttons and menus. However, only a few systems provide design guidance for their appropriate use (for example, Flanagan, et. al 1987; Frey and Wiederholt 1986). Several intelligent interfaces have also been reported which do not provide editing facilities or design advice, but use some type of rule base to select or tailor presentations using sets of display primitives (for example, Mackinlay 1986; Foley 1988). We review UIMS literature and intelligent interface literature as it relates to interface primitive support in Sections 4.4.1 and 4.4.2 respectively.

4.4.1 UIMS-based Libraries and Editors

Primitives editors and libraries, when compared with pixel editors, provide greater flexibility in editing and reasoning about displays, and they can reduce the effort required to integrate them with applications, but most do so by trading-off the designer's flexibility in creating them. They vary in terms of their ease of use. Some provide a library that requires the user to program in a conventional language (for example, Apple Computer 1985-1987) and others exploit direct-manipulation techniques to create direct-manipulation interfaces (Myers and Buxton 1986). KADD (Frey and Wiederholt 1986), LUIS (Lockheed 1987), CHIPS (Corbett 1986), DESIGNER (Weitzman 1986), Wlisp (Fischer 1987), and Peridot (Myers, and Buxton 1986) are examples of systems which cover a range of approaches with regard to the provision of display primitive libraries.

KADD (Frey and Wiederholt 1986) provides a set of seven high-level display elements for aircraft cockpit applications that it can evaluate with respect to designer defined requirements of the application, human factors considerations, and the con.istency of their use. The elements its supports are horizontal linear scales, vertical linear scales, circular scales, bar charts, column charts, digital readouts and status indicators, each requiring approximately 30 parameter values to be defined for a complete specification. It also supports an unspecified library of static graphic primitives. It does not produce functional interface software but provides a display driver to simulate application (aircraft) and interface behavior.

LUIS (Lockheed 1987) is a UIMS that provides a library of interface primitive objects that can be accessed and applied to applications running in either UNIX or Aegis environments. The library of objects are accessed and instantiated through either a high-level command language or menu-based editor, while the creation of new primitive objects is substantially more difficult. The system provides no guidance for the use of the primitives but incorporates data collection routines for post-design user evaluations using simulated or actual application software, and a parser that produces a text description of the objects comprising an interface.

CHIPS (Corbett 1986) provides an object-oriented library of primitives similar in scope to LUIS that can be used for applications which run in the LOOPS/Interlisp-D environment. It provides direct-manipulation access to certain object parameters, such as the positioning of objects on the screen, and the pop-editing of the driving functions and data.

DESIGNER (Weitzman 1986) incorporates an object-oriented library of display object "icons" tailored to the domain of the STEAMER mathematical model of a steam generation plant (Hollan, Hutchins, and Weitzman 1984). It is intended to support the generation of directly manipulable "views" of the simulation using this library and an intelligent design "assistant." This assistant operates to analyze, critique, or propose the use of primitives and the specification of their parameters, such as, color, size, and shape, based on the selection of a predefined set of constraints called a"style." The designer can either select a "style" such as simple, complex, or regular, and get help with the construction of a consistent presentation in the form of critiques and proposals for alternatives or have the system evaluate an existing design against alternative styles using an embedded truth maintenance system.

Wlisp (Fischer 1987), is another object-oriented interface design environment that is knowledge-based and runs on VAX systems using FranzLisp and a custom object-oriented knowledge representation language called ObjTalk. In addition to graphical primitives such as charts or graphs, its library includes such things as menus and windows. The system is based on a reuse approach to interface development, much like the worst-case innovation design model presented in Chapter 3, and on the philosophy that sophisticated direct-manipulation interfaces cannot be developed in parallel with an application, but must be an integral part of it. It is based on the view that the main activity of programming is not the creation of new programs but, rather, it is the redesign of existing ones and the reuse of building blocks. It makes extensive use of inheritance to reduce the need for specifying redundant information and to enable the creation of objects that are almost like other objects with a few incremental changes. The object oriented architecture currently supports more than 200 interface object classes which Fischer reports offer the following advantages over other interface design support architectures.

- The creation of subclasses of existing classes allows the designer to create new objects that differ from existing objects in some ways but that inherit almost all of the functionality of their ancestors.

- The use of predefined components is not an all-or-nothing decision. If a component has one undesirable property, the designer is not forced to abandon it completely. Even if most of the behavior of a superclass is undesired, the designer can still use it by overwriting all but the useful properties.

- Subclasses are not independent copies of their superclasses. They benefit from any augmentations of their superclasses.

- Extensions can be made on different levels of the hierarchy, thereby affecting selected classes of objects.

- Each method can be a hook for modifying behavior. Methods can be augmented by adding procedures to be executed before, after, or instead of existing methods.

Experience with Wlisp has indicated that a user requires considerable programming expertise and knowledge of the system before its full potential can be realized. It requires an extended period of learning and experimentation for this knowledge to be acquired. Several construction kits have been developed to support particular problems that are less complex (Fischer, Lemke, and Schwab, 1985).

In the literature surveyed, the Peridot system (Myers and Buxton 1986) takes perhaps the most ambitious approach to interface primitive use. The object of the research is to allow a designer to create interaction techniques by using examples of what the screen should look like, and what the user should do to perform an interaction. The system is highly experimental, however, and currently supports only a few examples to demonstrate the concept of designing by example.

4.4.2 Intelligent Interfaces

Intelligent Interfaces are designed to automatically adapt presentations to the context of a user interacting with an application. They are not intended to support the design of user interfaces but are relevant to the discussion of interface design tools because many intelligent interface architectures incorporate rules for formatting presentations, generating icons, or selecting from a fixed library of display primitives as a function of user behavior, task, domain, and application models. The demands that these systems place on their architectures are often less severe than those placed on design tools. This is because the models and rules they support are constructed for single applications and they run noninteractively, whereas the objective for a design tool is to support the reuse and construction of many such models for multiple designs and to incorporate designer involvement.

Like the UIMS research, graphic intelligent interface work was begun before direct-manipulation interfaces were widely available and was based on the assumption that user interface software should be separate from application software. The goal of automating the construction of interfaces to data bases and knowledge bases provided the impetus for some of the earliest work in this area. For example, the AIPS system (Zdybel, Greenfield, Yonke and Gibbons 1981), was one of the earliest attempts to separate the presentation process from the rest of the application. This system used a KL-ONE representation to specify and refine a high-level specification of a 2-D display. It automatically chose display formats from a fixed library to display the contents of a knowledge-base.The VIEW system (Friedell 1984) synthesized formatted multiple icons in response to data base queries, using knowledge of the domain entities in the data base and their relation to each other. This work extended the work initiated in the AIPS project by incorporating a more general computational mechanism for synthesizing graphical elements into meaningful object descriptions.

Other work in this area has focussed on the determination and construction of displays based primarily on data characteristics, and, to lesser extents, the task it is supporting and the domain in which it is being carried out. An early example of a system derived from this work is BHARAT (Gnanamgari 1981). BHARAT automatically selected chart and graph formats, that is, bar charts and line graphs, based on the attributes of the data being presented. For example, fractional quantities that summed to one were portrayed by a pie chart.

APT (Mackinlay 1986) is perhaps the most knowledge intensive system developed for automatic design, composition and rendering of chart and graphs from a set of low level graphic elements. The design is carried out by a synthesis algorithm that uses effectiveness criteria derived from the graphic perception task efficiency hierarchy proposed by Cleveland and McGill (1984) and Bertin's methods of graphic design (1983). The algorithm uses logic programming and a depth first search strategy with simple backtracking. It is capable of generating two dimensional static representations which require approximately two minutes to design and one minute to render. The list of ranked perceptual tasks used by APT to assess graph effectiveness as a function of the measurement scale of the data to be displayed are presented in Table 4.2. They were derived from Cleveland and McGill (1984) but have not been verified.

Rank		Coding Scheme	
	Quantitative Data	Ordinal Data	Nominal Data
1	Position	Position	Postion
2	Length	Density	Color hue
3	Angle	Color saturation	Texture
4	Slope	Color hue	Connection
5	Area	Texture	Containment
6	Volume	Connection	Density
7	Density	Containment	Color saturation
8	Color saturation	Length	Shape
9	Color hue	Angle	Length
10	Texture	Slope	Angle
11	Connection	Area	Slope
12	Containment	Volume	Area
13	Shape	Shape	Volume

Table 4.2 MacKinlay's (1986) ranking of graphic coding schemes expanded from Cleveland and McGill's (1984) work with quantitative data to cover ordinal and nominal data.

Other Intelligent interfaces are reported in the literature (for example, Arens et al. 1988; Tyler 1988) that take a more knowledge-based approach to intelligent interface design but the available descriptions do not provide information concerning the methods that they use to relate domain, task, and user characteristics to presentation configuration.

4.5 Aggregation of Primitive Elements into Screen Presentations

Support for the design of user interfaces that consider the relations among multiple display elements within and across presentations is provided by the Display Analysis Program developed by Tullis (1986), Designer (Weizenbaum 1986), KADD (Frey and Weiderholt 1986), and RIPL (Flanagan et al. 1987). The Display Analysis Program is a static screen presentation evaluator that is applicable to only alphanumeric displays. Designer is a design environment restricted to STEAMER applications. KADD is a knowledge-based design system developed for aircraft cockpit applications, and RIPL is a relatively shallow but comprehensive, general interface prototyping environment.

The Display Analysis Program (Tullis 1986) is a commercial product that runs on an IBM-PC and is, by far, the most comprehensive presentation evaluation tool reported in the literature. It takes literal examples of existing alphanumeric presentations represented as ASCII files, evaluates them against six evaluation criteria, and predicts visual search times and subjective evaluations using empirically estimated regression models. To make these predictions, it derives measurements of the overall density of the screen, the local density, the number of visual groups, the average visual angle subtended by each group, and the overall complexity of the layout. It also provides the user with diagnostic displays for each of the parameters listed and guidance on the appropriate ways to improve the display. The guidance is cast in the form of rules derived from a thorough review of the visual perception and human factors design literature (Tullis 1981, Tullis 1983) and the study of a sample of 520 displays that were evaluated by experimental subjects (Tullis 1984). Many of the recommendations are based on specific deviations from norms associated with the displays that were in the sample.

RIPL (Flanagan et al. 1987) is an interface prototyping environment that supports the production of dynamic facades, prototypes that simulate the look and behavior of a functional prototype but do not have functional software that can be integrated with an application. The system consists of five tools: a graphics editor, a simulator, a metric evaluator, an online technical library, and a knowledge-based help facility of very limited coverage. A RIPL interface facade is built from sets of linked "tile objects," that the user specifies in terms of text based attribute lists and a visual presentation produced by importing bit-mapped graphics from an Apple Macintosh or by using a set of drawing tools. For a given dialogue simulation, the metric evaluator can review the attributes of the component tiles and perform limited consistency and screen density checking on an inter- and intra-tile basis. What is measured is the consistent location and sizing of tiles defined to be of the same usage type, consistent stimuli sets for tiles defined to be of the same usage type, and measures of screen density for all possible state conditions of all possible screen presentations. The consistency measures are static in that they consider consistency within the set of like usage tiles and are insensitive to the dynamics of the facade. The screen density metric is state dependent and based on all possible combinations of presentations.

Designer (Weizenbaum 1986) couples a STEAMER domain knowledge base consisting of display design elements, design relations, and techniques for their identification, with sets of constraints that establish a context or style of presentation. The system uses these constraints to support the designer through an "Analyzer" that parses a design, based on display elements and relationships and records this information in a knowledge base. It also has a "Critiquer" that uses the products of the "Analyzer" along with the user defined style constraints to indicate where a design satisfies the constraints of a particular style, and a "Synthesizer," that proposes alternatives that satisfy style constraints. Designer creates an evaluation comment for each unsatisfied constraint within the current style. Descriptions and justifications of the comments are presented to the user in terms of the underlying principle or standard, which, unfortunately, are not well defined in the document describing the design. The design decisions that are made by the synthesizer are stored along with the comments from the critiquing phase. Alternative designs are maintained using an ATMS.

Frey and Weiderholt (1986) describe the Display Design Expert System component of KADD as composed of rules that evaluate displays for information requirement satisfaction and for consistency. In addition to checking that the ranges and precisions of individual display elements meet the related information requirements, it also checks aggregations of display elements associated with a particular task (a presentation) to verify that all of the information requirements for that task are satisfied by the presentation's sub-elements. Frey and Weiderholt (1986) also discuss the problems associated with representing global presentations in a manner that is sufficient for making evaluations that involve multiple display elements, and report that KADD can critique a presentation for consistency among display elements. However, Frey and Weiderholt do not describe exactly what factors KADD considers when checking for consistency.

4.6 Selection of Presentation Selection Methods

Several of the systems provide menu utilities and windows for organizing and managing multipresentation interfaces, but none of the systems provides design guidance for their use or methods for evaluating multipresentation designs.

There is a growing body of research concerned with aids for navigating through complex presentation structures (Billingsley 1981; Apperley, Field, and Waikato, 1985; Kaster and Widdel 1985), and ways to help users maintain their problem solving context as they transition from presentation to presentation (Woods 1984). This body of research is also concerned with the most effective ways to control and lay out multiple window presentations (Gaylin 1986; Bly and Rosenberg 1986; Card and Henderson 1987; Norman, Weldon and Schneiderman 1986) but the results of this research are not reflected in the systems we have reviewed. There is a version of the Xerox PARC ROOMS window system (Card and Henderson 1987) available for applications that run on Sun workstations, Xerox 1186, and other D machines (Tello 1988), but we have no knowledge of its availability or maturity at this time.

4.7 Prototype Development

Many of the systems we have reviewed are designed to support the construction of functional interface prototypes. They vary, however, in terms of the level of effort required to develop them, their run-time performance, the range of domains that they can support, the styles of dialogues that they can be used for, and the extent to which they support data collection. Overall, there seems to be a tradeoff between generality and useability, and design support appears to trade off with run time performance. At this time, there are no specific systems that can support the range of design functions we have described, and even as a group, the coverage of these systems is still relatively superficial. LUIS (Lockheed 1987), and CHIPS (Corbett 1986) support the development of limited but functional direct-manipulation interfaces. COUSIN/SPICE (Hayes 1985) can be used to develop interfaces to applications that require alphanumeric, form-based dialogues. Designer (Weitzman 1986) supports aspects of the design of direct-manipulation interfaces in addition to their implementation, but only for the STEAMER simulation program. KADD (Frey and Wiederholt 1986) appears to be one of the most comprehensive design environments, but the displays it produces are oriented to cockpit applications and as a consequence are nonfunctional in their native environment. RAPID/USE (Wasserman and Shewmake 1985) supports prototypes that simulate the look and feel of a design as does RIPL (Flanagan et al. 1987), but their products are not intended to be used with actual applications.

Many of the systems we have reviewed are experimental. However, there are a number of commercially marketed systems designed to support the development of fully functional, run time interfaces and prototypes (see Appendix C). In spite of their commercial availability, these systems are not used widely. In a recent review, Myers (1987) identifies three general problems associated with the current generation of commercial systems which he believes contribute to their lack of popularity. The first is that they are very hard to use and often require designers to learn a special purpose programming language. The second is that most of them are fairly limited in the types of interfaces they can be used to create, and the third is that the software produced by most of them is not portable.

A major reason for developing these interface design tools has been to make the development of interfaces easier. While there are exceptions, most of the available systems use a textual specification with a rigid syntax as complicated as a programming language. Some systems, like the Apple MacApp (Schmucker 1986) provide a framework for using a library of standard interface objects, but they still have to be programmed by conventional methods. Others require a user to learn a specialized language ranging from abstract BNF-like definitions to more declarative specifications (Hayes 1985). Only the most advanced systems, which are still experimental, allow designers to use visual programming languages (Jacob 1985) or direct-manipulation techniques for interface specification (Myers and Buxton 1986). Figure 4.4 presents various methods used to specify interfaces by the current generation of interface development systems, ordered in terms of the level of programming they require.

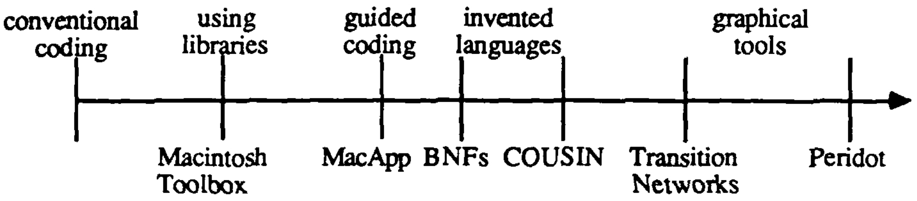


Figure 4.2 Methods for creating user interfaces ordered from left to right by ease of development (adapted from Myers 1987).

The second major problem with most of the available interface development tools is that they do not address direct-manipulation interfaces, or they do so in a very restricted way that compromises their performance. Some support them adequately, but only for specific restricted applications by providing hand-coded libraries of interface primitives for particular domains, applications, or computers (eg., Weitzman 1986). The Peridot system uses a programming by example paradigm (Myers and Buxton 1986) to support direct-manipulation, and Thinglab (Borning and Duisberg 1986) works within a constraint-based paradigm, but these systems are still being researched.

The third problem is that most interface development tools are tightly tied to a particular operating system, computer, or graphics package and consequently generate nonportable code. Myers (1987) explains this problem as a function of two things. One is that there are no appropriate high level interfaces that can shield the interface software from directly having to manage input devices such as the mouse and keyboard. Also there is no high-level interface that will not require them to directly create the screen graphics. The other reason is that the existing graphics standards (e.g. CORE, GKS, PHIGS) are slow, or too large to fit easily into many computers and, more importantly, the input model in all standards is the same and is widely perceived as wrong for highly interactive or direct-manipulation interfaces. These need rapid feedback to be successful.

4.8 Summary

In this chapter, more than twenty systems designed to support the design, development, and automatic generation of user-computer interfaces were reviewed in terms of the best-case design model presented in Chapter 3. Only three of these systems use AI technology to support the interface design with the remainder emphasizing interface implementation.

The developers of RIPL attempted to provide design support by generating a rule base from the human factors literature and found it to be one of the most difficult aspects of developing the system. They found they needed expert designer knowledge to supplement the literature, and that the rules that were generated were very complex and domain dependent. They also found that large amounts of problem definition data were required to be specified for each application before the system could provide design assistance. These issues and the fact that a 180 rule knowledge base was required to cover a very narrow aspect of the interface design domain led the RIPL developers to question the feasibility of expanding its coverage (Lenorovitz and Reaux 1986).

KADD developers also found the human factors literature inadequate for direct encoding into a rule base, and they found through designer evaluations that basing the system's design on a prescriptive, human factors design process model might preclude its use in the field because of its incompatibility with current practice. Designers reported a design process model that more closely mirrored the innovative, worst-case design model presented in Chapter 3, and expressed a preference for a system that emphasized tools that would enable them to quickly tailor and reuse existing designs rather than tools that would support a full information, unique design for every new application (Hunt and Frey 1987). Wlisp is a system developed explicitly to support the reuse and tailoring of existing designs through the exploitation of an object oriented inheritance architecture, but its developers report that the system is so complicated to use that most designers are unable to fully utilize the flexibility it provides (Fischer 1987) .

The User Interface Management Systems that have been developed and marketed commercially have experienced limited acceptance for a number of different reasons. The state-of-the-art with respect to the types of interfaces that can be developed using them lags behind the new direct-manipulation interface technology from which most applications appear to benefit. These current systems are difficult to use and the software they produce is often not portable to environments different from the ones in which they were developed. The demands for rapid feedback of direct-manipulation interfaces is raising questions concerning the separation of the interface software from the application software and I/O devices. This separation was a fundamental premise that the development of these systems were based upon (Myers 1987; Lowgren 1988).

In general, the presentation design tools that have been developed to date have shown that many of the premises that their designs were based upon are not viable. Human factors literature is as inadequate as the literature of other disciplines when it comes to using it to develop knowledge-based systems. It is useful for orientation but must be supplemented with considerable formal knowledge engineering before it can begin to approximate the expertise that it is based upon. The specific nature of the interface design problem is that it requires a substantial amount of domain, application, and task information for good design. This makes it difficult to provide general purpose guidance, and most designers indicate that organizational constraints often preclude the acquisition of adequate amounts of this information before design decisions have to be made. Because of this, questions are raised about the value of a knowledge-based design system that would require all these classes of information be available to the designers, when it typically is not.

Recent evaluations of the current generation of UIMS have pointed to the need for them to support the new generation of direct-manipulation interface technologies, and for them to support the incorporation of domain and task specific knowledge into the design process. However, it appears that this will have to be accompanied by changes in interface design conventions which typically do not allow an adequate schedule or resources to acquire and apply this information to the overall interactive system development process.

5. Proposals for An Intelligent Tool For The Design of Presentations

In this Chapter, we discuss, the functional goals and architectural issues surrounding the design of an intelligent tool for presentation design. This discussion is placed in the context of the constraints offered by the complexity and diversity of knowledge necessary to design a good presentation, and the strengths and limitations of the expert system approach to capturing this knowledge and expertise in a computer. Whether or not one undertakes to build a comprehensive design tool or a more limited system focused on some of the specialized subtasks that arise during the design process, an overall picture of the tool is an important guide to development. The architectural framework should convincingly support the variety of functions the tool must eventually perform. Moreover, the construction of large general tools progresses through stages, and the general framework has a valuable organizing influence on the order and division of effort among the subsystems. Even if one only builds a limited tool, one can profitably view it as a specific instance or component of the more comprehensive tool. In this way the limited tool serves to validate and refine the design for the more comprehensive tool.

5.1 Functional Design Goals

The best-case interface design model presented in Section 3.2.1 is an idealization of the process a user follows when designing an interface. It provides a useful picture of the tasks performed by a presentation designer—the domain, choices, and decisions he or she must make, and the sources of information on which to base those decisions. To automate the entire process lies beyond our ability today; a more realistic approach is to build a tool to aid the human designer, and therefore to ask ourselves how best to split the overall task between the human and the machine. The task determines the functional characteristics of any tool built to aid the process.

A notable property of the best-case interface design model is the number and variety of knowledge sources influencing the design of an interface. Some facts and guidelines are well-known, others obscure; some are generally accepted, others controversial; some are derived from formal definitions, others stated informally; some are well-described and available from many sources, others are vague and will require a concerted effort to acquire; some have a major effect on the finished design, others affect only minor details. This variety suggests a richly articulated system of many loosely connected components, which share a common representation language and are organized according to the origin and focus of the knowledge they imbed.

The variety of knowledge sources also implies that the tool, like a human expert, will have its strengths and weaknesses, and therefore must allow a mixed-initiative interaction in which the user and the system freely collaborate on the finished design. The natural v ay to express the design knowledge will also vary from source to source and from application to application. Sometimes knowledge is best expressed as a critique of an existing design, whereas other times knowledge can be expressed as a proposal for a new design element. Certainly many shadings of meaning are possible between *proposing* and *evaluating* a design. These modes of expression will give rise to different dialogues between the user and the tool; suggesting, specifying, critiquing, and so on. These will be discussed more fully in Section 5.3.

Some important general properties of the interface design task[1] were apparent in the work reported in Section 4.1, and need to be incorporated in any proposal for a presentation design tool.

- The interface should be based on established paradigms. Ideally, these paradigms represent the distillation of huge amounts of practical experience, and implicitly embed considerable human factors expertise. Their use is the most reliable route to ensuring that desirable global properties of the interface are met: uniformity, learnability, robustness. Interface paradigms are embodied in primitives for carrying out the various interactions between the interface and its users. An appropriately chosen set of primitives will range over small (e.g., how to solicit typed input) and large (e.g., spreadsheet manipulations) interactions.

- The interface should require a minimum of extraneous specification. Specification can be extraneous by virtue of being simply repetitive, or redundant in that previous choices obviously restrict or completely determine a new property of the interface. To achieve this, the tool's primitives must be explicitly represented internally, their known properties and constraints on their values or their relationships to other properties must be explicitly stated.

- The tool should support innovative design and the reuse of existing interface fragments. As our review of the practice of interface design presented in Chapter 3 has shown, it is rare to design something from scratch. Just as a well chosen set of primitives provides building blocks, prior work provides a wealth of prototypes to be incorporated into a new design.

[1] These properties are no doubt equally valid for any design task.

Referring back to the best-case interface design model, we observe that one of the most important transitions occurs between the DEFINE REQUIREMENTS and DESIGN INTERFACE stages. The need to accommodate information about a user's task (and, implicitly, the domain), however indirectly, in the process is the greatest challenge faced by the presentation design tool. Many of the important decisions made during the design of an interface can be made only with reference to an understanding of what the eventual user of the interface needs to accomplish. Questions of proper sequencing of window configurations, the availability of information and guidance when it is needed, the choice of a presentation or interaction paradigm, for example, can be made only in the explicit context provided by a model of the task. Practically speaking, many visual characteristics of an interface (e.g., colors, shapes, display types) are determined—for better or worse—by the conventions of the domain or task.

A major contribution of the presentation tool to the work of the interface designer is to shift his or her focus away from the myriad details of how the interface looks towards how useable it is. This is entirely appropriate, since the nominal interface designer knows most about the uses to which the interface will be put. A danger confronting the presentation tool builder is that the ultimate burden on the tool's users may not be lessened, but rather shifted from needing to specify too many details about the appearance of the screen to needing to specify a similarly excessive amount of information about the task. Moreover, a user must perceive a clear and immediate advantage to specifying task information if the tool is to be well received. We propose a design in which salient characteristics of the task are collected implicitly in the course of explicitly constructing a portion of the interface.[2]

5.2 Architectural Issues

The principal functional design goals discussed in the previous section suggest a set of architectural requirements for an intelligent presentation design tool. We present these requirements as part of a comprehensive overall design for the tool that will support the critical aspects of both the best- and worst-case design models, but recognize that subsequent discussion will select and refine those portions actually implemented.

The thrust of the tool is to help its users (interface designers) bridge the gap between a task-oriented view of the interface and a presentation-oriented view. Figure 5.1 depicts this as

[2] One might call it "situated task knowledge acquisition," to coin a somewhat ponderous phrase.

specifying an element of the task and transforming it into a specification in terms of some underlying set of directly realizable interface primitives.[3]

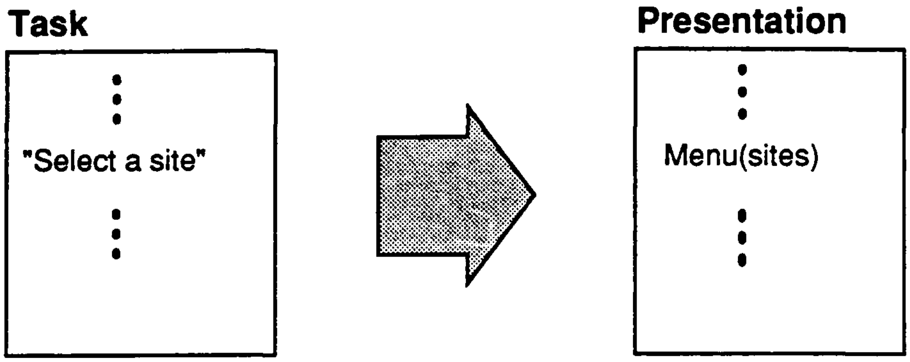


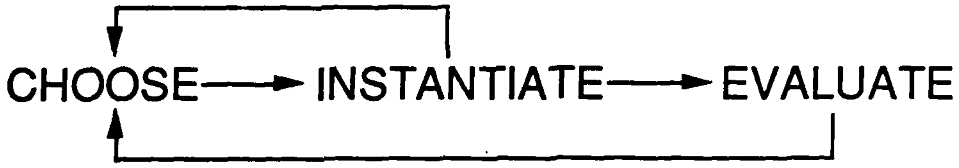
Figure 5.1 The design tool supports two views of the interface—Task and Presentation—and translations between them.

For example, if one imagines a context of building an interface to some battle management application, where one of the actions required of the user is to "Select a site," the tool helps translate this specification into a command to put up a menu of relevant sites. "Select a site" is one of many task primitives available to the designer. Note that it implicitly incorporates some domain knowledge—the notion of a "site".

There are other ways to translate this specification. Instead, one might want to sensitize all the sites visible on a map, and let a user select a site simply by pointing to it with a mouse; or prompt a user for a name of a site; or some combination of these techniques. Additionally, every choice brings with it other attributes that affect the ultimate appearance of the presentation. In the case of a simple menu of sites, there are still the issues of where to place it on the screen, how to format it, how to label it, what fonts to use, how to prompt the user, whether multiple choices are allowed, and what sites are included, to name a few.

[3] We do not intend to restrict users to a task-oriented specification of the interface, but expect them to interact directly with both the Task and Presentation views of the interface design. The mark of a successful tool will be that it makes specification via Task view (the best-case model) so easy and effective that it becomes the method of choice for building interfaces.

The primary cycle of interaction with the tool can be depicted as follows:



The designer and the tool work together to create two interlinked descriptions of the interface: the Task View and the Presentation View. In either view, a choice is made from available primitives[4] to specify some unit of interaction, which leads to further choices to specify related properties of the interaction or components of the interaction, and to an evaluation and possible reconsideration of those choices. The design tool's expertise is distributed among these functions in the form of Consultants, conceptual agents that encapsulate particular types of knowledge about presentation design and apply it in the form of choices or evaluations.

5.2.1 Design Consultants

Consultants are the principal organizing software construct in the design tool. They are triggered by events in the design process, or by features introduced into a design either in the presentation view or task view, and represent the source of the tool's involvement in the design process. Consultants have unique identities. They may contribute to any of the choice, instantiation, or evaluation phases of the design process, and may play any of a variety of different roles with respect to how assertively they attempt to influence design decisions.

Consultants may be specialists or generalists. Some might focus on isolated features of the presentation (e.g., whenever a user is prompted for information, some technique must be used to ensure that his or her attention is drawn to it). Others might focus on more overarching features of the design (e.g., whether color is used consistently to represent attributes of the domain). One imagines writing consultants to deal with issues of color, shape, screen layout, legibility of text, choice of graphs, label placement, highlighting style, maps, symbols, continuity from screen to screen, for example. Because general design guidelines often conflict, still other consultants might play the role of "adjudicators," mediating the choice between conflicting suggestions or evaluations.

[4] The use of the term "primitive" does not mean to necessarily imply simplicity, but rather that the designer is choosing from a predefined collection of interaction elements and assemblages.

Consultants differ also in their granularity; that is, the autonomy of their behavior and scope of their decisions. While some consultants may consist of but a few rules, at the other extreme a consultant can be a complex program in its own right, prepared to take control of an entire subdialogue in the design process. For example, the layout of a diagram consists of so many interconnected decisions that the overall supervision might be allocated to a single consultant, a specialist that oversees the specification. Another example is the specialist that chooses a type of graph, based on properties of the information to be shown. In the middle of this granularity spectrum, consultants make more modest contributions to the design, drawing on other consultants to share in the decisions. For example, the placement of a symbol on a map involves the choice of color, position, and label, decisions that can be delegated to other consultants, including the consultant that remembers the decisions on past symbols and offers these in the name of continuity.

Why allow such diversity among the consultants? Because, as the reviews reported in Section 3 make clear, design knowledge in the community occurs in many forms and any attempt to incorporate that knowledge in an intelligent system requires great flexibility in how it can be expressed. The knowledge acquisition activity involved in building any intelligent system is formidable, but is alleviated by having in the system the same abundant set of constructs that practitioners spontaneously use to couch their knowledge.

Consultants have a <u>focus</u>; that is, the aspect of the design about which they have something to say. Equally important, consultants have a <u>context</u> in which they are activated. The context ranges over all features of the design. An individual consultant's initial applicability and eventual contribution to the design depends on the domain, the task, the presentation being built, output device capabilities, the application, the current design state, and the designer's input. All these are part of the world of the consultant and can affect its behavior. Two crucial elements of this context are provided by taxonomies of tasks and of presentations.

5.2.2 Task and Presentation Taxonomies

Taxonomies of task fragments and presentation types are the backbone of the design tool. They create an internal language by which descriptions of tasks are linked to corresponding presentations, and by which consultants are classified. The taxonomies also establish the language[5] through which a designer specifies the task and presentation he or she wants to see.

Figure 5.2 shows a fragment of the task taxonomy. The taxonomy should be viewed as a semantic net in which important classes and relationships are represented. At the top of the taxonomy several important abstractions appear that are shared by more specific concepts below it; for example, the direction of information flow and interdependencies among parts of a task. In the middle region of the taxonomy important interaction paradigms are introduced, which precisely capture a purely information-oriented description of potential interactions between a user and an application. At the bottom of the taxonomy these paradigms have been specialized to cover actual elements in the task domain.

By building domain- and task-specific elements into the taxonomy, the tool allows the designer to pursue a task-oriented approach to building an interface. Knowledge of the task is acquired by the tool in the course of specifying the sequence of presentations; more information may be solicited by the tool while instantiating a task fragment and more context is available to aid the consultants.

A presentation-oriented approach to building an interface is still available to the designer through the presentation taxonomy shown in Figure 5.3.

Like the task taxonomy, the presentation taxonomy reveals several general concepts and relations near the top and increasingly specific derived concepts below. But unlike the task taxonomy, the presentation taxonomy explicitly embeds knowledge about the appearance of information on the screen. The most abstract classes pertain to the modality by which information is conveyed, and whether it is being recognized or produced by the system. The middle layers introduce general components, features, and relationships shared by many different kinds of displays. At the bottom, specific types of information displays and means of gathering input from a user appear. These are the actual implementation level building blocks for the interface. A

[5] The language implies a system of reference, a set of objects and relationships between them, used to identify elements in the task and presentation views. We do not wish to raise the specter of "natural language" as the preferred means of communication between the user and the design tool.

presentation-oriented description of the interface can be turned into a working interface by realizing the device-specific variants of these lowest level primitives.

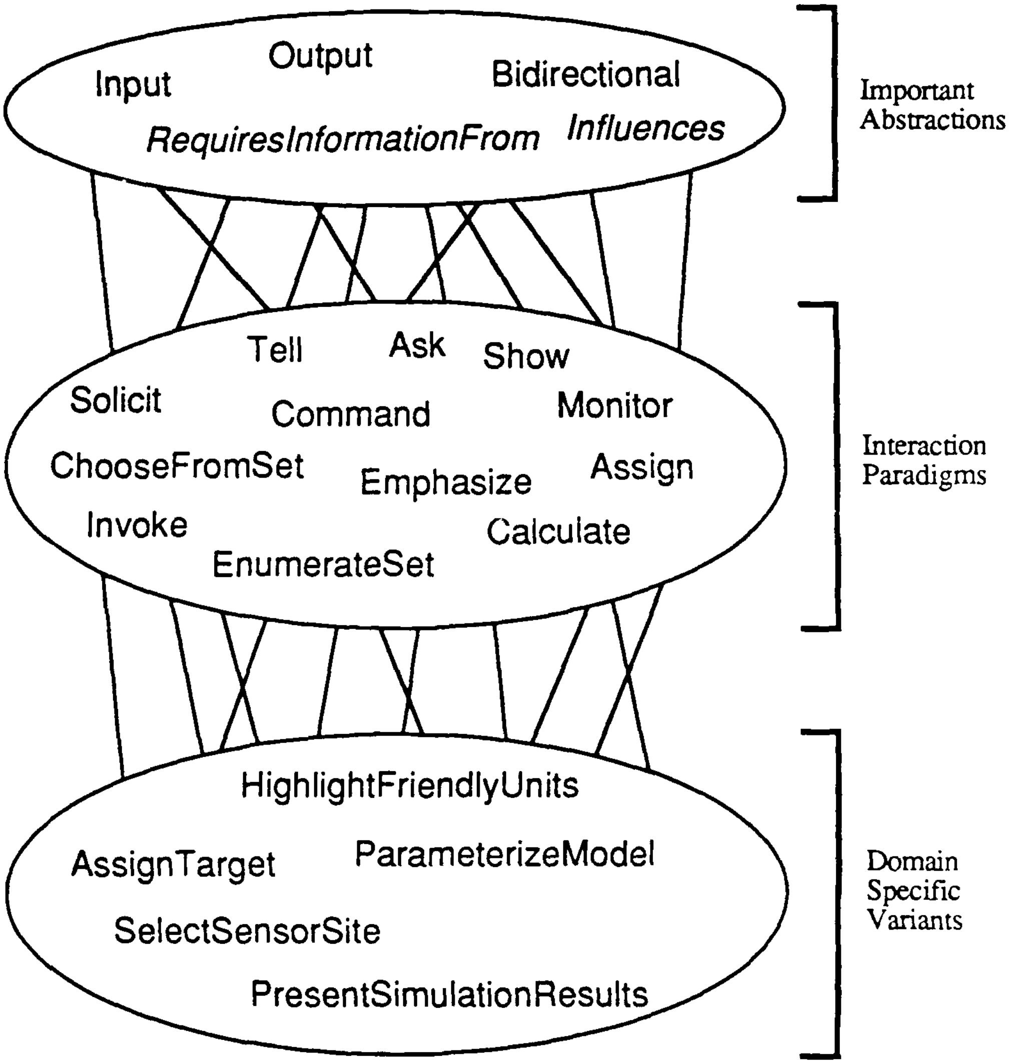


Figure 5.2 A portion of the task taxonomy. Concepts are shown in normal typeface; relations are italicized.

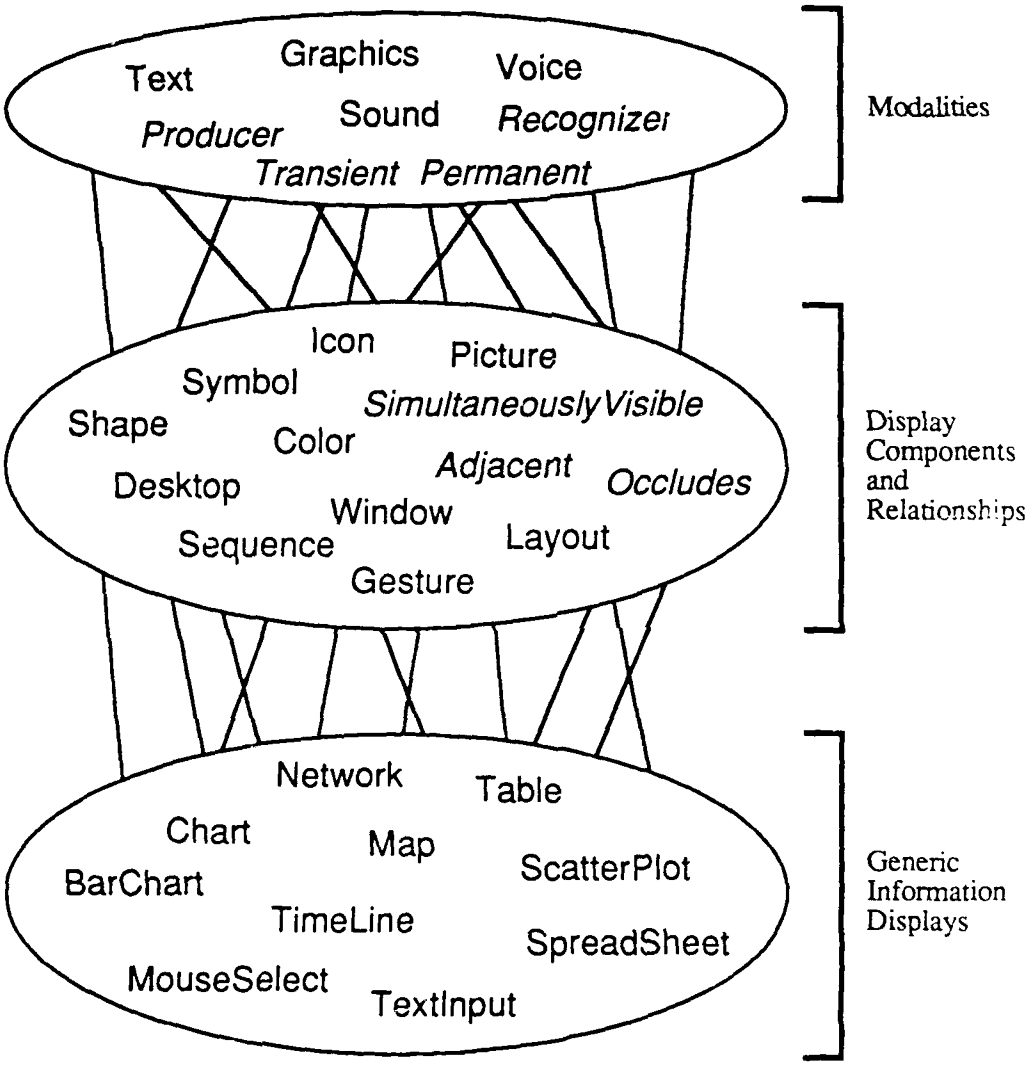


Figure 5.3 A portion of the presentation taxonomy. Concepts are shown in normal typeface; relations are italicized.

5.2.3 Knowledge Representation Issues

In this section, the discussion returns to issues surrounding the writing of consultants: namely, the kinds of knowledge structures that need to be provided and the kinds of reasoning that need to be supported by the design tool.[6]

Taxonomic representations of objects and relations have already been introduced in Section 5.2.2. In addition, consultants rely on knowledge expressed in the following forms:

- Patterns and Specifications. A pattern language provides a consistent mechanism for referring to objects, their features, and relationships. It can equally well refer to objects singly or as sets, and can identify objects indirectly by their properties or associations. The same pattern language can be used to assert facts about objects; i.e., specifications. Viewing the context through the eyes of a common pattern language makes writing consultants simpler since one does not have to be continually aware of the exact underlying representation, which may well differ for different parts of the context.

- Structure Mappings. These explicitly represent correspondences between the Task and Presentation Views of the interface by associating features across this representational boundary. In general mappings are not simple one-to-one relations, but rather capture multiple associations. The results of a consultant's work is often best expressed as a mapping. For example, a convention for allocating a visual feature of a display to showing some property of a set of domain objects (e.g., Color = Nationality) is easily represented as a structure mapping.

The major categories of graphical presentations (maps, diagrams, networks) will have associated consultants to oversee their instantiation. Instantiating a presentation is a knowledge-intensive activity, since one must supply sufficient detail to transform a schema for the presentation into a specific description. The choice of detail must be sensitive to properties of the information the presentation has been chosen to convey as well as make appropriate use of the visual features of the display.

A framework for constructing these consultants is described in Appendix A, which includes an extended example to illustrate the approach. Briefly, the framework introduces

[6] Knowledge structures and reasoning mechanisms are obviously intimately related.

principles for defining an *Interpretive Mapping* from semantic features of the information onto visual features of the display.

• Rule Sets. Certainly consultants will make widespread use of rule-based knowledge—this is a prevalent form of expert expression but in order to be tractable, related rules must be organized into functional groups. Experience has shown that rules expressing design guidelines are extremely context-dependent. Consequently, before rules can be succcssfully applied it must be established that they are relevant to the current situation. Collecting rules that have similar contexts into rule sets wili make consultants easier to write and more efficient to use.

• Constraint Sets. Much design knowledge concerns reciprocal relations among features of the interface, which are best expressed as constraints. Like rules, constraints are context dependent and benefit from being organized into functional groups. Within a set of constraints other relations can be made explicit that are useful for resolving constraints by adjusting, weakening, ranking, or ignoring them.

Based on these knowledge structures, several forms of reasoning are necessary to support the styles of interaction a designer expects to have with the sign tool. They are well understood AI techniques and familiar components of many intelligent systems.

• Taxonomic Inferences. Much of the value of the task and presentation hierarchies stems from the relative ease with which some useful inferences can be made within them. Generalization/specialization, part/whole, and before/after are all relations computable from the explicit taxonomic representation, and whose use is widespread among descriptions of tasks and presentations.

• Forward Chaining. A typical use of rules is to reveal the consequences of a design decision. By chaining from the new facts engendered by the decision to a larger set of implied facts, these rules reformulate the consequences of a decision in terms more easily recognized by the relevant consultants.

• Backward Chaining. Rules can also express how to make choices to meet a requirement by reasoning backward from a description of a desired state of affairs to antecedents known to produce it. The antecedents then become the focus of attention for furthering the design. Backward chaining rules represent "how to" knowledge, and are useful for generating mappings from a task into its presentation.

- Constraint Relaxation. General mechanisms for satisfying multiple complex constraints are not powerful enough to satisfy the needs of the design tool. However, the natural form of expression allowed by stating design knowledge in the form of constraints makes them an attractive adjunct to the reasoning of the system. So, rather than rely on general mechanisms, the tool expects additional explicit knowledge to be associated with individual constraints to aid in the constraint satisfaction process. For a single constraint, this knowledge takes the form of methods for perturbing the design in a direction anticipated to meet the constraint. By grouping constraints into identifiable sets, knowledge can be associated with a set that likewise proposes design changes anticipated to satisfy simultaneous interacting constraints.

Guidelines often appear as preferences for maximizing or minimizing some parameter of the design; constraints and their associated methods for altering the design provide the means of representing this knowledge in the tool. The use of constraints to state desirable characteristics of a design demands mechanisms not just for altering the design to satisfy a constraint, but also for adjusting the constraint to satisfy a design. The wealth of guidelines and opinions about interface design, even with a good mechanism for defining contexts and controlling the applicability of constraints, guarantees conflict among otherwise sound constraints. To deal with this situation, a mechanism is needed to moderate individual constraints and to adjudicate between multiple constraints. Explicit knowledge can be associated with a constraint for weakening or otherwise adjusting its parameters, thereby preserving the intention of the constraint at a level consistent with competing constraints on the design. Constraints can also point to alternates to be used instead of the primary one; "third party" constraints get invoked to completely take over a decision, or to act as arbitrator, when two constraints are in conflict.

- Dependency Maintenance. Because design necessarily involves trial and error, the design tool supports backtracking from decisions made about the appearance of the interface. It relies on dependency maintenance to locate and choose among prior decisions. The reasons for a decision, in the form of the user specifications, consultants, constraints, or rules determining it, are remembered in the dependency network, and can be used to assess accountability for factors leading up to any point in the design. The dependencies provide a memory of the decisions and the assumptions that produced the finished interface.

5.3 User Interface Requirements

The previous proposals for functional goals of an intelligent presentation design tool and the architectural consequences of realizing these goals fail to adequately describe the appearance of the system from the designer's point of view. This section examines the following question: How is the tool's interface going to look and what is it going to feel like to use?

5.3.1 The Editing Environment

The presentation design tool must be an editor. However significant its own contribution to the finished design, the tool must allow the designer complete control over the finished interface. By providing a means of specifying the design not only in terms of its appearance, but in the terms of the tasks to be performed with it, the tool helps build more appropriate, more useable interfaces. But it is still an aid to the designer, not a replacement.

Thinking about the tool partly as an editor suggests a critical feature of its interface. It must provide a means of *seeing* the object of the design process. The tool provides two views—Task and Presentation—and both must be accessible to the user if he or she is to intelligently specify and modify the design. Seeing the Task View helps the designer assess the ability of the interface to support the information flow required by the task; e.g., whether information is available on the screen or accessible via commands to permit an informed response by the user. The Presentation View helps the designer assess the visual and aural consequences of the design.

The Task View presentation, so to speak, is well shown graphically as a network of information paths, decision points, procedure invocations, and subtask decompositions. A corresponding textual version would read more like a hypothetical script to be followed by the user.

Deciding on a Presentation View presentation raises the interesting question of how precisely one needs to see the actual implemented interface to judge it effectiveness, at least during the construction phase of the design. The problems of producing a working interface from an incomplete design are formidable, and the surprising thought occurs that the interface itself may not be its own best presentation. An alternate view that makes apparent more of the internal conclusions of the design process would better serve the needs of the designer; for example, to see that a symbol is not just "red," but the color of the opposing forces, or to see that not just a particular prompt, but all prompts will appear at a single screen location. This alternate view of the visual aspects of a design could be captured in a caricature [illegible] interface, an annotated paste-up of the screen at any point in the task, where the annotation records hidden facts about visible details of the screen.

Any view of the design is not just a static record, but a canvas on which consultants, constraints, and rules show their results, present alternatives, and make proposals. Having a good presentation for the thing being edited allows direct-manipulation of its elements, and therefore the designer's primary means of access to the system is through the Task and Presentation Views. The Cut–Copy–Paste–Insert metaphor for editing operations accurately describe the major interactions the designer has with the tool.

The Task and Presentation Views are not the only windows onto the inner state of the design tool. The reasoning components of the system, the consultants, can also benefit from a graphical presentation of their own. In general, the focus of the design, which consultants are active, the constraint adjudication process, and actions available to the designer all require presentations.

5.3.2 Mixed Initiative Interaction

The consultants built into the design tool are the source of its own initiative; they actively propose, guide, and comment on the work of the human designer. This leads to a mixed initiative style of interaction with the system in which both the user and the system are able to choose the next step and freely respond to the actions of the other. The personality of the system is that of an intelligent, but obedient, assistant.

To achieve this style of interaction, some simple discourse prerequisites need to be met: first, the focus of the interaction must be explicit, and second, a means of directing attention to a new part of the design must be available to both the user and the system. For the user, the intended focus is implicit in the commands and directives issued to the tool (e.g., pick a color, create a choice, show an object). The focus is an important part of the context that selects and guides the consultants. Unlike a human counterpart, the system continually displays its notion of the focus, so there is no ambiguity about how a command was interpreted.[7] In addition, the system constantly displays to the user what actions it would perform in the current situation, but as suggestions that are easy to invoke rather than being imposed on the user.

5.3.3 Dialogue Types

The designer manipulates the design directly via the editors associated with the Task and Presentation Views, and indirectly by invoking consultants. Consultants are categorized

[7] A problematic aspect of human discourse is that one must infer from their subsequent actions and responses the understanding of an utterance by its listener. Explicitly and dynamically showing this element the internal state of a system is an attempt to avoid this problem in human-computer discourse.

according to the role they are prepared to play in the design process; namely, how much initiative they bring to bear. These differences reflect the fact that any particular aspect of design knowledge may be expressed more easily in one mode than another. Each of these roles evokes a different style of dialogue with the user.

A spectrum of dialogue types is shown below:

COMMENT → GUIDE → CRITIQUE → PROPOSE

The dialogue types increase in assertiveness from left to right. Commentary is used to provide feedback on user's actions, acknowledging decisions and tracing the system's responses, and to provide a continuously updated statement of the system's assessment of the design, for example, deferred decisions and underspecified presentations. Guidance provides the user with suggestions for what action to take next, what kind of choices are desirable, and the constraints on allowable values. Guidance can be in the form of passive statements read by the user, or applied constraints that actively limit the choices available to the user. Criticism results from evaluating choices and specifications, whether made by the system or the user. Negative criticism takes the form of an objection, identifying the offending property; positive criticism adds to this suggestions for remedial action. Proposals represent the system's own choice.

One can see how these different forms of expression arise from the consultant's reliance on rule sets, constraint sets, and the other knowledge structures, and how the reasoning capabilities support the various dialogues. For example, criticism depends on running rules that recognize undesirable design decisions and then uses the constraints to suggest improvements. Proposals and guidance use a combination of forward and backward chaining to suggest choices or actions.

To a lesser extent, the user initiates a similar range of dialogues with the system. User commentary includes establishing the focus (what part of the task to work on next, for example) and stating preferences, in the form of rankings between constraints, or choosing which consultants to activate. Rejecting a system proposal, and directing the backtracking mechanism to produce an alternate is the user's way of criticizing the system. Outright specifications are proposals, the user's primary mode of expression.

The term dialogue is appropriate for describing these interactions because they do not consist of a single exchange between the user and the system, but rather a coherent sequence of exchanges. For example, the user invokes an evaluation of some aspect of the design; the system responds with a critique, say a constraint is violated; the user asks to weaken the constraint; the system proposes a design change; the user accepts it. These dialogues share important components, or subdialogues, which appear frequently in the course of interaction: explanation—the immediate source of the system's statements, elaboration—presenting more detail about the chain of reasoning leading up to a decision, negotiation—to handle constraint resolution, exploration—to browse through the system's design knowledge.

6. Recommendations for Future Efforts

Previous sections of this report have reviewed the extensive literature on interface design, presented a framework for classifying and evaluating this work, summarized the state of the art in terms of the tools that have been implemented to aid the design process, reviewed the relevant expert system approaches to building more intelligent systems, and proposed a set of functional goals and their associated architectural issues for an intelligent tool for the design of presentations. In this final section, we recommend a series of steps and alternatives for future development.

The proposals of section 5 attempt to be comprehensive in their coverage of issues, and can be best understood as a framework for guiding future work. A realistic program of development is going to choose among the themes presented—pursuing some immediately, and deferring others until later in the program. The first set of recommendations is modest in that it builds maximally on existing work and is oriented toward near term solutions. The second pair of recommendations is more aggressive in dealing with a broader set of problems.

6.1 Task-Specific Interface Paradigms

One approach to improving the quality of interfaces is to improve the components out of which they are built—by providing a consistent set of well-designed primitives. Furthermore, by taking into account specifics of the tasks which the interfaces will serve, an even more carefully tailored set of primitives can be offered.

Many classes of human factors guidelines and conventions can be incorporated implicitly in an interface by constraining its designer to a standardized set of building blocks, but making them easy to use and combine. However, for this approach to succeed, the selection has to be rich enough to satisfy the intentions of the designer, and must not be limited to just the simpler forms of interaction (menus, commands, text input, buttons, and so on) but include higher level constructs as well (editing a table, selecting and placing elements on a map, modifying interlinked parameters, and the like).

Consequently, one recommendation is to develop and implement one or more of these higher level interaction paradigms, focusing particularly on the needs of Battle Management, Command and Control, or any other task domain that is particularly relevant to the user community. A set of standards, even informal ones, can be a straightjacket if they are not well chosen. Having available a powerful set of interface paradigms to build into new and existing interfaces would dramatically improve their usability.

An analysis of the tasks would help reveal points of maximum leverage for improving the usability of an interface and guide the choice of paradigm, but in this recommendation no task description is incorporated explicitly into the finished system. This recommendation recognizes that some interface limitations mu^st be overcome by developing entirely different kinds of paradigms than can be assembled from available techniques, and that the intent of guidelines is best served by an entirely new form of interaction.[8] An extreme example would be a system that could produce sound and voice in addition to printing and drawing on a screen, in which relegating alerting functions to the voice channel removes a whole set of issues about how to interrupt and direct attention of the user by purely visual means.

A more ambitious version of this recommendation is to implement an entire interface, that is, work on a complete solution for an application, but do so with an ey^ toward general solutions. Develop, in a specific but representative context, the reusable parts and conventions that will make subsequent interfaces for the same class of application easier to build and higher quality. This recommendation recognizes that one shouldn't build tools in isolation from their eventual use. Moreover, developing well-abstracted interaction primitives is an evolutionary process—it's necessary to use them to refine them properly.

For this interface building enterprise to be most fruitful in the long term, one must take evaluation seriously—not just collect anecdotal evidence in response to its use. It is necessary to find objective measures for evaluating an interface in a realistic setting, and to extract the reasons underlying each individual's responses and performance using a system.

Finally, one must build on available interface techniques and code. The emphasis should be on extending, not replacing, existing interface "toolkits". The level of effort involved in constructing a set of interface building primitives is too great to not maximally exploit the state of the art. A carefully chosen commercial package, or community supported set of standards, is the right starting point.

[8] It is an unavoidable consequence of interface development and human factors research that guidelines are sought and then described in terms of available techniques. Guidelines tend to be written in terms specific to a particular style of interaction, and often make implicit assumptions about available hardware or software capabilities. Consequently, it may be difficult or misleading to attempt to apply them to situations not foreseen at the time the guidelines were conceived.

6.2 Knowledge Acquisition for Intelligent Presentations

A second recommendation is to focus on a central problem of building a presentation design tool—incorporating enough knowledge of the task to genuinely improve the ability of the interface to support the work of the user. The first step is to acquire sufficient task and presentation knowledge to build the taxonomies described in Section 5. These are the basis for any further work in developing a presentation design tool. Naturally, a representative collection of tasks would be chosen, in order to constrain the investigation. The goal of the investigation would be to determine the descriptors that make up the intermediate and high level abstractions in the task taxonomy, and to link them to the primitive task elements. Recall that the purpose of this task taxonomy is to represent the actions taken by an application and a user, as well as the information exchange present in the task. It does not however include any references to the visual aspects of the presentation.

Contrary to what has been proposed elsewhere (Williges et al. 1987; Carter 1986 Lenorovitz et al. 1984; Cohill 1984), it is our firm belief that one cannot build the task taxonomy in isolation. The eventual goal is the mapping from the task taxonomy to the presentation taxonomy, associating a task–oriented description of an interaction with a corresponding presentation–oriented description. With this in mind, the two taxonomies should be built together. A set of interface primitives provides the starting point for the presentation taxonomy, but work would still be needed to determine the best higher level descriptors that ensure the mappings are general. By working from examples of good interfaces for known tasks and characteristic task fragments, some optimal mappings can be made by hand. These direct the knowledge acquisition process. As a side–effect of this process, the adequacy of presentation primitives is tested. If no suitable presentation can be found for a task, this suggests that a new primitive is needed.

Experience shows that knowledge acquisition should be undertaken with a clear sense how that knowledge is going to be used. A model of the eventual functioning of the system guides the acquisition process by focusing attention on relevant aspects of a domain (i.e., those that matter to the reasoning processes underlying the system), and by constraining how the knowledge is represented. Therefore, the effort to acquire taxonomic knowledge of tasks and presentations must be supplemented by building some of what in section 5.2.1 were called *consultants*.

6.3 Intelligent Interface Consultants

The large number and variety of consultants imagined as part of a mature presentation design tool demand that some pragmatic choices be made to determine where to begin. To aid these decisions, it is useful to classify the world of consultants, and to establish one's needs along the various dimensions that distinguish them.

One place to restrict development effort is to limit the domain and the kinds of tasks for which interfaces are to be designed. For example, within battle management are found several categories of tasks (plan, execute, monitor, simulate, instruct) and domains (logistics, communications, intelligence, weather) that can provide a restricted focus for early efforts.

Another dimension is the type of presentation paradigms to be addressed. One might choose to focus on the specialized interactions one has with maps, tables, diagrams, or plots — when to choose them, how to parameterize them for a particular context, and how to structure the user interaction.

Still another dimension is the type of human factor's concern. For example, one might build a color consultant to aid in the consistent choice of appropriate colors, or a consultant to evaluate the legibility of text material, or the availability to the user of information about the status of his or her interaction with an application.

Consultants were described as belonging to different dialogue types with respec .o how actively they participated in the design process. Initial development might focus on only one of these types; for example, the critical evaluation of completed designs.

However one chooses to limit the tool's early development, the intent should be to evolve its capabilities as dictated by deficiencies apparent under conditions of actual use. Section 6.1 touches on the requirements of evaluating interfaces; they are applicable here as well.

6.4 Intelligent Interfaces

One final recommendation for a future direction stems from the observation that the knowledge and capabilities applicable to the design of presentations could equally wel' be applied within the interface itself to make it more responsive and adaptable (see for example, Becker, Cleve'^nd and Wilks 1987; Cleveland, McGill, and McGill 1988). One might investigate how to apply this same knowledge to improving the behavior of interfaces. Incorporating these capabilities directly in the interface is another way to validate and refine techniques for choosing presentations appropriate to the current task, ensuring the ready availability of information to the user when it is needed, supplying missing details that specify how to construct a display, and other features of an intelligent interface. There is sufficient overlap between the intelligent functioning and design of interfaces that working within both contexts simultaneously would be mutually beneficial. In fact, for many capabilities, exploring them within the context provided by a real working interface would greatly accelerate their development due to more visible and immediate feedback.

Of course, not all decisions about the functioning of an interface can be deferred until runtime! The computational demands and informational requirements of making the right choices will always push some of the decisions into the design stage. On the other hand, research is showing that more and more decisions are best left until the last moment (Cleveland, McGill, and McGill 1988). Consider the decision about the shape parameter of a two variable graph when the detection of changes in trends are of interest. The best choice might depend on the particular distributions of each variable and their relationship, which would not be known precisely ahead of time. The interface designer can deal only with questions or purpose that the data will serve, and the general characterizations of the data, not the actual values.

Other problems are exacerbated by deferring interface decisions until runtime; for example, incorporating information about the task in the choices. If the interface is not to interfere with the user's communication with some underlying applications, it must try to track the user's progress through the task, inferring intentions from his or her actions. The interface designer, however, has the time and the flexibility to specify important aspects of the task directly, so that they can be used by the design tool in its reasoning.

Because of the overlapping concerns of intelligent interface design and function, it seems profitable to explore them together. In either case, a successful program should be grounded in a particular domain and task set, and allow for iterative development, realistic evaluation and reimplementation.

7. References

ACM/SIGGRAPH,"Bibliography of Software Tools for User Interface Development," Computer Graphics, Vol. 21, No. 2, 1987, pp.145-147.

Adelson, B., and E. Soloway, "A Model of Software Design," Research Report 342, Department of Computer Science, Yale University, October, 1984.

Andriole S. J., and S. M. Halpin, "Information Technology for Command and Control: Introduction to Special Issue on Information Technology for Command and Control," IEEE Transactions on Systems, Man, and Cybernetics, November/December 1986, Vol. SMC-16. No. 6, pp.762-765.

Apple Computer Inc., Inside Macintosh, Reading, MA: Addison-Wesley, Vol. 1-5, 1985-1987.

Apple Computer Inc., Apple Human Interface Guidelines: The Apple Desktop Interface, Reading MA: Addison-Wesley, 1987.

Apperley, M. D., Field, and Waikato," A Comparative Evaluation of Menu-Based Interactive Human-Computer Dialogue Techniques," in Human-Computer Interaction-INTERACT '84, B. Shakel (Ed.) North Holland, 1985, pp.323-328.

Arens, Y., Miller, L., Shapiro, S. C., and N. K., Sondheimer, "Automatic Construction of User-Interface Displays," paper presented at The 1988 Conference of the American Association of Artificial Intelligence, 1988, St. Paul, MN, August 22-26.

Bachant, J., and J. McDermitt, "R1 Revised: Four Years in the Trenches," The AI Magazine, Vol. 5, No. 3, Fall 1984.

Banks, J., and J. S. Carson II, Discrete-Event System Simulation, Prentice-Hall: Englewood Cliffs NJ, 1984.

Batini, C., E. Nardelli, and R. Tamassia, "A Layout Algorithm for Data Flow Diagrams," IEEE Transactions on Software Engineering, Vol. SE-12, No. 4, April 1986.

Beach, R., and M. Stone, "Graphical Style: Toward High Quality Illustrations," Computer Graphics, 1983, Vol.17, No. 3, pp.127-135.

Becker, R. A., Cleveland, W. S. and A. R. Wilks, "Dynamic Graphics for Data Analysis," Statistical Science, Vol. 2, No. 4, pp.355-395.

Bennett, J. L., "Collaboration of UIMS Designers and Human Factors Specialists," Computer Graphics, Vol. 21, No. 2, 1987, pp.102-105.

Bertin, J., Semiology of Graphics: Diagrams, Networks, Maps , (translated by W. J. Berg), University of Wisconson Press: Madison,WI, 1983.

Betts, W., Burlingame, D., Fischer, G., Foley, J., Green, M., Kasik, D., Kerr, S. T., Olsen, D., and J. Thomas, "Goals and Objectives for User Interface Software." Computer Graphics, Vol. 21, No. 2, 1987, pp.73-78.

Billingsley, P., "Navigation Through Hierarchical Menu Structures: Does it Help to Have a Map?" Proceedings of the Human Factors Society 26th Annual Meeting, 1982, pp.103-107.

Bly, S. A., and Rosenberg, J. K., "A Comparison of Tiled and Overlapping Windows," Proceedings of CHI '8 , April 1986, pp.101-106.

Boar, B. H., Application Prototyping: A Requirements Definition Strategy for the '80s, New York: Wiley, 1984.

Bobrow, D. G., Mittal, S. and M. J. Stefik, "Expert Systems: Perils and Promise," Communications of the ACM, September 1986, Vol. 29, No. 9, pp.880-894.

Boose, J., and B. Gaines, "Special Issues: Knowledge Acquisition for Knowledge-based Systems." Parts1-5, International Journal for Man-machine Studies, Vol. 26, nos. 1, 2, 4, 1986; Vol. 27, nos. 2, 3, 1987.

Borning, A., and R. Duisberg, "Constraint-Based Tools for Building User Interfaces," ACM Transactions on Graphics, 1986, Vol. 5, No. 4, pp.345-374.

Brown, D. C. and B. Chandrasekaran, "Plan Selection in Design Problem Solving," Proceedings of the AISB 85 Conference on Artificial Intelligence, Warwick UK, April 1985.

Buchanan, B., Barstow, D., Betchtel, R., Bennett, J., Clancey, W., Kulikowski, C., Mitchell, T., and D. A. Waterman, "Constructing an Expert System," in Building Expert Systems, F. Hayes-Roth, D. A. Waterman and D. B. Lenat Eds. Addison -Wesley: Reading MA 1983.

Bury, K. F., "Windowing versus Scrolling on a Visual Display Terminals," Human Factors, Vol. 24, No. 4. pp.85-88.

Card, S. K., "A Multiple, Virtual-Workspace Interface to Support User Task Switching," Proceedings of CHI+GI 1987, pp.53-59.

Card, S. K., Moran, T. P., and A. Newell, The Psychology of Human Computer Interaction, Hillsdale NJ: Lawrence Earlbaum, 1983.

Carroll, J. M., Thomas, J. C., and Malhotra, A. "Clinical-Experimental Analysis of Design Problem Solving," Design Studies , Vol. 1, No. 2, October 1979, pp.84-92.

Carroll, J. M. Thomas, J. C., and Malhotra, A., "Presentation and Representation in Design Problem-Solving," British Journal of Psychology, Vol. 71, 1980, pp.143-153.

Carter, J. A., "A Taxonomy of User-oriented Functions," International Journal of Man-machine Studies, 1986, Vol. 24, pp.195-292.

Chapanis, A., "The Relevance of Physiological and Psychological Criteria to Man-machine Systems: The Present State-of-the-Art," Ergonomics 1970, Vol. 13, No. 3, pp.337-346.

Chapanis, A., "Interactive Human Communications," Scientific American, 1975, Vol. 232, No. 3, pp.36-42.

Chambers, J. M., Cleveland, W. S., Kleiner, B., and P. A.Tukey, Graphical Methods For Data Analysis, Belmont CA: Wadsworth, 1983.

Chiba, N., Onoguchi, K., and T. Nishizeki, "Drawing Plane Graphics Nicely," Acta Informatica, Vol. 22, 1985, pp.187-201.

Chignell, M. H., Chol, D., Peterson, J. G., and G. Nadler, "The Validity of Rational Approaches to Conceptual Design," in Proceeding of the Annual Meeting of the IEEE Systems, Man and Cybernetics, 1987, pp.995-998.

Cleveland , W. S., "Research in Statistical Graphics," Journal of the American Statistical Association, Vol. 82, No. 398, 1987, pp.419-423.

Cleveland , W. S., Harris, C., and R. McGill, "Experiments on Quantitative Judgements of Graphs and Maps," Bell Systems Technical Journal Vol. 62, 1983, pp.1659-1674.

Cleveland , W. S., and R. McGill, "An Experiment in Graphical Perception," International Journal of Man-machine Studies, Vol. 25, 1986, pp.491-500.

Cleveland, W. S., and R. McGill, "Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods," Journal of the American Statistical Association Vol. 79, No. 387, 1984, pp.531-554.

Cleveland , W. S., and R. McGill, "Graphical Perception and Graphical Methods for Analyzing Scientific Data," Science, Vol. 229, August 1985, pp.828-833.

Cleveland, W. S. "Research in Statistical Graphics" Journal of the American Statistical Association , Vol. 82, No. 398, 1987, pp.419-423.

Cleveland, W. S., McGill, , and R. McGill, "The Shape Parameter of a Two-Variable Graph" Journal of the American Statistical Association Vol. 83, No. 402, 1988, pp.289-299.

Cleveland, W. S., and M.E.McGill,, Dynamic Graphics for Statistics, Monterey, CA: Wadsworth, in press.

Cohen, R. M., May, J. H., and H. E. Pople, "A Study of Expert Decision-Making in Design" Proceedings of the 1986 IEEE International Conference on Systems, Man, and Cybernetics, Piscataway, NJ: Institute of Electrical Engineers, pp.1325-1330.

Cohill, L. F., "A Taxonomy of User-computer Interface Functions," in G. Salvendy, Ed., Human Computer Interaction, 1984, Amsterdam: Elsevier, pp.125-128.

Corbett, J. D., Chips: A Toolkit for Editing Interfaces: Programmer's Guide, Pittsburg PA: Learning Research and Development Center, University of Pittsburg, 1986.

Corker, K., Davis, L., Papazian, B., and R. Pew, "Development of an Advanced Task Analysis Methodology and Demonstration for Army-NASA Aircrew/Aircraft Integration," BBN Report No. 6124, Bolt Beranek and Newman Inc., 1986.

Davis, A. M., "A Comparison of Techniques for the Specification of External System Behavior," Communications of the ACM, 1988, Vol. 31, No. 9, pp.1098-1115.

Dumais, S. T., and W.P. Jones, "A Comparison of Symbolic and Spacial Filing," Human Factors in Computing Systems, April 1985, pp.127-130.

Dunn, R. "Framed Rectangle Charts or Statistical Maps With Shading: An Experiment in Graphical Perception," The American Statistician, Vol. 42, No. 2, pp.123-129.

Duffy, A., "Bibliography-Artificial Intelligence in Design," The International Journal for Artificial Intelligence in Engineering, Vol. 2, No. 3, 1987, pp.173-179.

Drury, C. G., Paramore, B., Van Cott, H. P., Grey, S. M., and E. N. Corlett, "Task Analysis," in G. Salvendy (Ed.) Handbook of Human Factors, 1987, New York: John Wiley & Sons, pp.370-401.

Eastman, C. M., Explorations of the Cognitive Processes in Design, Department of Computer Science, Carnegie-Mellon University, Pittsburg PA, 1968.

Elkerton, J., and R. C. Williges, "A Performance Profile Methodology for Implementing Assistance and Instruction in Computer-based Tasks," International Journal of Man-Machine Studies, 1985, Vol. 23, pp.135-151.

Ericsson, K. A., and H. A. Simon, Protocol Analysis and Verbal Reports as Data, 1984, Cambridge: MIT Press.

Feiner, S., "APEX: An Experiment in the Automated Creation of Pictorial Explanations," IEEE CG&A, November 1985, pp.29-37.

Ferrante, R. D., and K. A. Tench, "TIDAS: An Interactive Man-Macine Interface Prototyping Environment," Proceedings of CHI '86, 1986, pp.1478-1483.

Finegold, A., "The Engineers Apprentice," in The AI Business, The Commercial Uses of Artificial Intelligence, P. H. Winston, and K. A. Prendergast (Eds.),1984, Cambridge MA: MIT Press, pp.67-119.

Fischer, G., "An Object-oriented Construction and Tool Kit for Human-Computer Interaction," Computer Graphics, Vol. 21, No. 2, 1987, pp 105-108 .

Fisher, H. T., Mapping Information, Cambridge MA: Abt Books, 1982.

Fitter, M.J. and T. R. G. "When Do Diagrams Make Good Computer Languages?" in M. J. Coombs and J. L. Alty (Eds.) Computing Skills and the User Interface,Academic Press, London 1981, pp 253-288.

Flanagan, D. P., Blue, D. V., Giacaglia, R. A., Lenorovitz, D. R., and E. C. Stanke, Rapid Intelligent Prototyping Laboratory (RIPL) Architecture and Use, Computer Technology Associates, 1987.

Foley, J., Gibbs, C., Kim, W. C., and S. Kovacevic, " A Knowledge-based User Interface Management System," CHI '88 Proceedings, 1988, pp.67-72.

Foley, J. D., and V. L. Wallace, " The Art of Natural Graphic Man-machine Conversation," 1974 Procceedings of IEEE, 1974, Vol. 63, pp.462-471.

Follettie, J. F., "Real-World Tasks of Statistical Graph Using and Analytical Tasks of Graphics Research," unpublished paper presented at The Annual Meeting of the National Computer Graphics Association, Anaheim CA, 1986.

Frey, P. R., and B. J. Wiederholt, "KADD-An Environment For Interactive Knowledge Aided Display Design," Proceedings of the IEEE Conference on Systems, Man and Cybernetics, 1986, pp.1472-1477.

Friedell, M., "Automatic Synthesis of Graphic Objects," Computer Graphics, Vol. 18, No. 3, July 1984, pp.53-62.

Gargan, R. A., Sullivan, J. W., and S. W. Tyler, "Multimodal Response Planning: An Adaptive Rule-Based Approach," Proceedings of CHI '88, pp.229-234.

Gaylin, K. B., "How are Windows Used?" Proceedings of CHI '86, pp.96-100.

Gnanamgari, S., Information Presentation Through Default Displays, PhD Dissertation, University of Pennsylvania, 1981.

Good, M. D., Whiteside, J. A., Wixon, D. R., and S. J. Jones, "Building a User-Defined Interface, Communications of the ACM, Vol. 27 No. 10, pp.1032-1043.

Goodman, D., The Complete Hypercard Handbook, New York: Bantam Books, 1987.

Gould, J., Boies, S., Levy, S., Richard, J., and J. Schonard, "The 1984 Olympic Message System: A Test of Behavioral Principles of System Design," Communications of the ACM, Vol. 30, No. 9, 1987.

Gould, J. D., and C. Lewis, "Designing for Usability-Key Principles and What .Designers Think," Proceedings CHI '83 Conference: Human Factors in Computing Systems, 1983, pp.50-53.

Gould. J. D., Conti, J., and T. Hovanyecz, "Composing Letters with a Simulated Listening Typewriter. Proceedings of the 25th Annual Meeting of the Human Factors Society, 1981, Santa Monica, CA: The Human Factors Society, pp.505-508.

Green, M., "The University of Alberta User Interface Management System," Computer Graphics, Vol. 19, No. 3, 1985.

Green, M., "A Survey of Three Dialogue Models," ACM Transactions on Graphics, Vol.5, No. 3, 1986, pp.244-275.

Green, M., "Directions for User Interface Management Systems Research," Computer Graphics, Vol. 21, No. 2, 1987, pp 113-116.

Green, T. R. G., Sime, M. E. and M. J. Fitter, "The Art of Notation," in M. J. Coombs and J. L. Alty (Eds.) Computing Skills and the User Interface, Academic Press, London 1981, pp 221-252.

Gross, M., and A. Fleisher, "Design as The Exploration of Constraints," DesignStudies, Vol. 3, No. 5, 1984, pp.137-138.

Green, P., and R. W. Pew, "Evaluating Pictographic Symbols: An Automotive Application,", Human Factors, 1978, Vol. 20, No. 4, pp.103-114.

Hammond, N., Jorgensen, A. MacLean, A. Barnard, P. and Long, J. "Design Practice and Interface Usability: Evidence from Interviews with Designers," Proceedings of CHI '83: Human Factors in Computing Systems, December 1983, pp.40-44.

Hayes, P., "Executable Interface Definitions Using Form-based Interface Abstractions," in Hartsen, H. (Ed.), Advances in Human-Computer Interaction, Ablex Publishing: NJ, 1985, pp.243-281.

Hayes-Roth, F., Waterman, D. A., and D. Lenat, "An Overview of Expert Systems," in Building Expert Systems, F. Hayes-Roth, D. A. Waterman and D. B. Lenat (Eds.) Addison-Wesley, Reading MA 1983.

Helander, M. G., "Design of Visual Displays," in G. Salvendy (Ed.) Handbook of Human Factors, 1987, New York: John Wiley & Sons.

Henderson, D. A., and S. K. Card, "Rooms: The Use of Multiple Virtual Workspaces to Reduce Space Contention in a Window-based Graphical User Interface," ACM Transactions on Graphics, Vol. 5, No. 3, 1986, pp.211-243.

Herring, B. E., and W. Prather, "A Simulation Model for Analyzing Service Times in a Rotational Position Seeking Disk System," Simulation, Vol. 46, No. 5, May 1986, pp.185-191.

Hiltz, S. R., and M. Turnoff, "The Evolution of User Behavior in a Computerized Conferencing System," Communications of the ACM, 1981, Vol. 24, pp.739-751.

Hollanagel, E., and D. D. Woods, "Cognitive System Engineering: New Wine in New Bottles," International Journal of Man-Machine Studies, Vol. 18,1983, pp.583-600.

Hollan, J. , Hutchins, E. L., and L. Weitzman, "STEAMER: An Advanced Computer-assisted Instruction System for Propulsion Engineering," Proceedings of the 1984 Summer Computer Simulation Conference, pp.400-404.

Hollands, J. , and P. M. Merikle, "Menu Organization and User Expertise in Information Search Tasks, Human Factors, 1987, Vol. 29, No. 5, pp.577-586.

Hunt, R. M., and P. R. Frey, "Knowledge Aided Display Design (KADD) System: An Evaluation," Proceedings of the Human Factors Society 31st Annual Meeting, 1987, pp.536-540.

Hutchins. E., Hollan, J., and D. A. Norman, "Direct-Manipulation Interfaces," in Norman, D. A., and S. Draper,(Eds.) User Centered System Design, Lawrence Erlbaum Associates: Hillsdale NJ, 1986, pp.87-124.

Hunt, R. M., and P. R. Frey, "Knowledge Aided Display Design (KADD) System: An Evaluation," Proceedings of the Human Factors Society-31st Annual Meeting, 1987, pp. 536-540.

Jacob, R., "An Executable Specification Technique for Describing Human Computer Interaction," in Hartsen, H. (Ed.), Advances in Human-Computer Interaction, Ablex Publishing: NJ, 1985.

Jacob, R., "A Specification Language for Direct-Manipulation Interfaces," ACM Transactions on Graphics, Vol. 5, No. 4, 1986.

Kahn, K. M., "Creation of Computer Animation from Story Descriptions," PhD Thesis, Massachusetts Institute of Technology, 1979.

Kaman, R., "The Comprehensibility of Printed Instructions and the Flowchart Alternative," Human Factors, Vol. 17, No. 2, 1975, pp.183-191.

Kant, E., "Understanding and Automating Algorithm Design," Proceedings of the Ninth International Joint Conference on Artificial Intelligence, August, 1985, pp.1243-1253.

Kantowitz, B. H, and R.D. Sorkin, Allocation of Functions," in G. Salvendy (Ed.) Handbook of Human Factors. 1987, New York: John Wiley & Sons, pp.355-369.

Kaster,J. and H.Widdel, "Graphical Support for Dialogue Transparency," in Human-Computer Interaction-INTERACT '84, B. Shakel (Ed.) North Holland, 1985, pp.329-333.

Kelly, J. F., "An Empirical Methodology for Writing User-Friendly Natural Language Computer Applications, Proceedings of Human Factors in Computing Systems, 1983, New York Association for Computing Machinery, pp.193-196.

Kosslyn, S. M., "Graphics and Human Information Processing," Journal of the American Statistical Association, Vol. 80, 1985 , pp.499-512.

Kosslyn, S. M., "Understanding Charts and Graphs" working paper 1988.

Laughery, K. R., and K. R. Laughery, "Analytic Techniques for Function Analysis," in G. Salvendy (Ed.) Handbook of Human Factors. New York: John Wiley & Sons, 1987.

Lenorowitz, D. R., Philips, M. D., Ardrey, R. S., and G. V. Kloster, "A Taxonomic Approach to Characterizing Human-computer Interfaces." in G. Salvendy Ed., Human Computer Interaction, Amsterdam: Elsevier, 1984, pp.111-116.

Lenorowitz, D. R., and R. A. Reaux, "Integration of Human Factors Guidance Information within the USI Design/Rapid Prototyping Process," in Proceedings of the 1986 IEEE National Aerospace and Electronics Conference, Dayton, OH: Institute of Electrical and Electronics Engineers, 1986, pp.926-934.

Lindsay, R. K., Buchanan, B. G., Feigenbaum, E. A., and J. Lenderber, Applications of Artificial Intelligence for Organic Chemistry. The DENDRAL project, Mac Graw Hill, 1980.

Lipton, R. J., North, S. C., and J. S. Sandberg, "A Method for Drawing Graphs," Proceedings of the Symposium on Computational Geometry, 1985, pp.153-160.

Lockheed Corporation, Lockheed User Interface System (LUIS) User Manual, Austin TX: Lockheed Austin Division, December 1987.

Lowgren, J., "History, State, and Future of User Interface Management Systems," SIGCHI Bulletin, Vol. 20, No. 1, 1988, pp.32-43.

MacGregor, A. J. Graphics Simplified, University of Toronto Press, Toronto, 1979.

Mackinlay, J., "Automating the Design of Graphical Presentations of Relational Information," ACM Transactions on Graphics, Vol. 5, No. 2, 1986, pp.110-141.

Maguire, M., "An Evaluaticn of Published Recommendations on the Design of Man-Computer Dialogues," International Journal of Man-Machine Studies, Vol. 16, 1982, pp. 237-261.

Malhotra, A., Thomas, J. C., Carroll, J. M. and Miller, L., A., "Cognitive Processes in Design," International Journal of Man-machine Studies, Vol. 12, 1980, pp.119-140.

Malone, T. W., "How Do People Organize Their Desks? Implications for the Design of Office Information Systems," ACM Transactions on Office Information Systems, 1983, Vol. 1, pp. 99-112.

Mancini, G, Woods, D. D., and E. Hollnagel (Eds.), "Special Issue: Cognitive Engineering in Dynamic Worlds," International Journal of Man-machine Studies, Vol. 27, No. 5&6, 1987.

Mantei, M. M. and Teorey, T. J., "Cost /Benefit Analysis for Incorporating Human Factors in the Software Life cycle," Communications of the ACM, Vol. 31, No. 4, April 1988, pp.428-439.

Marcus, A., "Computer-Assisted Chart Making from the Graphic Designers Perspective," Computer Graphics (Proceedings of SIGGRAPH '80), Vol. 14, No. 3, 1980, pp.247-253.

McDermott, J., "R1: A Rule-Based Configurer of Computer Systems," Artificial Intelligence Vol.19, No. 1, 1982, pp.39-88.

Meister, D., "Systems Design, Development, and Testing," in G. Salvendy (Ed.) Handbook of Human Factors. 1987, New York: John Wiley & Sons, pp.17-42.

Meister, D., Human Factors: Theory and Practice, New York: Wiley, 1971.

Meister, D., Behavioral Analysis and Measurement Methods, New York: Wiley, 1985.

Meister, D., and Farr, D. E. "The Utilization of Human Factors Information by Designers," Human Factors, Vol. 9, No. 1, 1967, pp.71-88.

Mitchell, T. M., Steinberg, L. I., and J. S. Shulman, A Knowledge-Based Approach to Design, IEEE Transactions on Pattern Analysis and Pattern Analysis and Machine Learning PAMI-7, Vol. 5 September, 1985, pp.502-510.

Mittal. S., Bobrow, D.G., and J. de Kleer, DARN: A Community Memory for A Diagnosis and Repair Task. Xerox ISL Lab. Report, Xerox Corporation, Palo Alto, CA, 1985.

Mittal, S., and Dym, C. L., "Knowledge Acquisition From Multiple Experts," The AI Magazine, Summer 1985, pp.32-36.

Mittal, S., Dym, C. L., and M. Morjaria, "Pride: An Expert System for the Design of Paper Handling Systems," in Applications of Knowledge-Based Systems to Engineering and Design, C. L. Dym, (Ed.) American Society of Mechanical Engineers, NEW YORK, 1985.

Moran, T., "Introduction to the Command Language Grammar: A Representation for the User Interface of Interactive Computer Systems," Xerox PARC Report SSL-78-3, AIP Memo 111, Palo Alto CA: Xerox PARC, 1978.

Morgan,C., Williams, G., and P. Lemmons, "An Interview with Wayne Rosing, Bruce Daniels, and Larry Tesler: A behind the scenes look at the development of Apple's Lisa," Byte Magazine, February 1983, pp.90-114.

Mostow, J., "Toward Better Models of the Design Process," AI Magazine, Vol. 6, No. 1, 1985, pp.44-57.

Myers, B. A., "Gaining General Acceptance for UIMS," Computer Graphics. Vol. 21, No. 2, 1987, pp.130-134.

Myers, B. A., and Buxton, W. "Creating Highly Interactive and Graphical User Interfaces by Demonstration," Computer Graphics, Vol. 20, No. 4, 1986, 249-258.

Nadler, G., and J. G. Peterson, "Successful Design in Organizations: Time line Scenarios," Submitted to the International Journal of Applied Engineering Education. 1987.

Norman, D. A, "Steps Toward Cognitive Engineering: Design Rules Based on Analysis of Human Error," 1983 Proceedings of Human Factors in Computing Systems, NEW YORK, Association of Computing Machinery, 1983, pp.378-382.

Norman, D. A., "Cognitive Engineering," in User-Centered System Design: New Perspectives on Human-Computer Interaction, D. A. Norman and S. W. Draper (Eds.) Lawrence Erlbaum: Hillsdale NJ, 1986.

Norman, D. A., Draper, S. W., and L. J. Bannon,"Glossary," in User-Centered System Design: New Perspectives on Human-Computer Interaction, D. A. Norman and S. W. Draper Eds. Lawrence Erlbaum Hillsdale NJ, 1986.

Norman, K. L., Weldon, L. J., and B. Shneiderman, "Cognitive Layouts of Windows and Multiple Screens for User Interfaces," International Journal of Man-Machine Studies, Vol. 25, 1986, pp.229-248.

Perrow, C., "The Organizational Context of Human Factors," Administrative Science Quarterly, Vol. 28, 1983, pp.521-541.

Peterson, J. G., Nadler, G., and M. Chignell, "Perspectives on Design: Problems, Processes and Purposes," in Proceeding of the Annual Meeting of the IEEE Systems, Man and Cybernetics, 1987, pp.980-984.

Pew, R. W., Baron, S., Feehrer, C. E., and D. C. Miller, "Critical Review and Analysis of Performance Models Applicable to Man-machine Systems Evaluation," BBN Report No. 3446, Bolt Beranek and Newman, Inc., Cambridge MA, 1977.

Philips, M. D., and K. Tischer, "Operations Concept Formation for Next Generation Air Traffic Control Systems," Proceedings of INTERACT '84 First Conference on Human Computer Interaction, 1984, London, pp.242-247.

Pinker, S., "A Theory of Graph Comprehension," Occasional Paper 10, MIT Center for Cognitive Science 1981.

Pinker, S., "Pattern Perception and Comprehension of Graphs", Report ED 237-339, MIT, Psychology Department, 1983.

Prerau, D. S., "Selection of an Appropriate Domain for an Expert System," The AI Magazine, Summer, 1985, pp.26-30.

Reisner, P., "Further Developments Toward Using Formal Grammar as a Design Tool," 1982 Proceedings of Human Factors in Computing Systems, New York: Association of Computing Systems, 1982, pp.304-308.

Rhyne, J., Ehrich, R., Bennett, J., Hewett, T., Sibert, J., and T. Blesser, "Tools and Methodology for User Interface Development," Computer Graphics, Vol. 21, No. 2, 1987, pp.78-87.

Rogers, J. G., and Armstrong, R., "Use of Human Factors Standards in Design," Human Factors Vol.19, No. 1, 1977, pp.15-24.

Rosson, M. B., Maass, S. and Kellogg, W. A., "Designing for Designers: An Analysis of Design Practice in the Real World," Proceedings of CHI '87: Human Factors in Computing Systems, pp.137-142.

Rouse, W. B., Systems Engineering Models of Human-machine Interaction, NEW YORK: North Holland, 1980.

Rouse, W. B., "Design and Evaluation of Computer-based Decision Support Systems," in G. Salvendy (Ed.), Human Computer Interaction, Amsterdam: Elsevier, 1984.

Rowe, L. A., Davis, M., Messinger, E., Meyer, C., Spirakis, and A. Tuan, "A Browser for Directed Graphs," Software-Practice and Experience,.1987, Vol.17, No. 1, pp.61-76.

Rubenstein, R., and Hersh, H. M., The Human Factor: Designing Computer Systems for People Burlington MA: Digital Press 1984.

Schmid, C. F., Statistical Graphics, NEW YORK: John Wiley, 1983.

Schmid, C. F., and S. E. Schmid, Handbook of Graphic Presentation, 2nd Ed., Wiley: New York, 1979.

Shafer, D., Hypertalk Programming, Indianapolis IN: Hayden Books, 1988.

Shortliffe, E. H., Computer-based Medical Consultations: MYCIN. New York: Elsevier, 1976.

Simkin, D., and Hastie, R., "An information-Processing Analysis of Graph Perception," Journal of the American Statistical Association, Vol. 82, No. 398, 1987, pp.454-465.

Simoes, L., and J. Marques, "IMAGES-An Object Oriented UIMS. Proceedings of INTERACT '87, Esevier Science Publishers, BV, 1987.

Smith, D. C., Irby, C., Kimball, Verplank, W., and E. Harslem, "Designing the Star User Interface," Byte Magazine, April 1982, pp.242-282.

Smith, S.L., and J. N. Mosier, Design Guidelines for User-System Interface Software, ESD Report #: ESD-TR-190 1984.

Stefik, M., Aikens, J., Balzer, R., Benoit, J., Birnbaum, L., Hayes-Roth, F., and E. Sacerdoti, "The Architecture of Expert Systems," in Building Expert Systems, F. Hayes-Roth, D. A. Waterman and D. B. Lenat (Eds.) Addison Wesley: Reading MA 1983.

Steinberg, L. I., "Design=Top Down Refinement Plus Constraint Propagation Plus What?" Proceedings of the 1987 IEEE International Conference on Systems, Man, and Cybernetics, Institute of Electrical Engineers, pp.498-502.

Swanston, M. T., and C. E. Walley, "Factors Affecting the Speed of Acquisition of Tabulated Information from Visual Displays," Ergonomics, Vol. 27, No. 3, 1984, pp.321-330.

Tello, E. R., "Windows," AI Expert, October 1988, pp.36-41.

Tong, C., "Toward an Engineering Science of Knowledge-based Design," The International Journal for Artificial Intelligence in Engineering, Vol. 2, No. 3, 1987, pp.133-166.

Tufte, E. R., The Viual Display of Quantitative Information, Cheshire CT: Graphic Press: Cheshire, Conn, 1983.

Tullis, T. S.,"An Evaluation of Alphanumeric, Graphic and Color Information Displays," Human Factors, Vol. 23, pp.541-550.

Tullis, T. S., "The Formatting of Alphanumeric Displays: A Review and Analysis," Human Factors, Vol. 25, pp.657-682.

Tullis, T. S., Predicting the Usability of Alphanumeric Displays, Ph.D. Dissertation, Rice University, 1984.

Tullis, T. S., Display Analysis Program (Version 4.0), Lawrence KN: The Report Store, 1986.

Tullis, T. S., "A System for Evaluation Screen Formats," Proceedings of the Human Factors Society 30th Annual Meeting, 1986, Santa Monica CA: Human Factors Society.

Tullis, T. S.,"Design of Effective Displays," Proceedings of USICON '87, Austin TX, February 29, 1987, pp.1-26.

Tyler, S. W., "SAUSI: A Knowledge-based Interface Architecture," Proceedings of CHI '88, pp.235-240.

Virtual Prototypes Inc., Virtual Prototyping Technology Information Package, 1987, Virtual Prototypes Inc.

Wainer, H. and Francolini, C. M. "An Empirical Inquiry Concerning Human Understanding of Two-Variable Color Maps," The American Statistician, 1980,Vol. 34, No. 2, pp.81-93.

Wainer, H. and Thissen, D. "Graphical Data Analysis," in Annual Review of Psychology Vol. 32, 1981, pp.191-241.

Wasserman, A. I., and D. T., Shewmake, "The Role of Prototypes in the User Software Engineering (USE) Methodology," in Hartsen, H. (Ed.), Advances in Human-Computer Interaction, Ablex Publishing: NJ, 1985, pp.191-210.

Waterman, D., A., A Guide to Expert Systems, Addison-Wesley: Reading MA, 1986.

Weitzman, L., Designer: A Knowledge-based Graphic Design Assistant Institute for Cognitive Science, Report ICS 8609, San Diego CA: University of California, July 1986.

Williams, G., "The Lisa Computer System: Apple Designs a New Kind of Machine," Byte Magazine, February 1983, pp.33-50.

Williges, R. C., and Williges, B. H., "Dialogue Design Considerations for interactive Computer Systems," in F. A. Muckler (Ed.) Human Factors Review 1984,Santa Monica CA: The Human Factors Society 1984, pp.167-208.

Williges, R. C., Williges, B. H. and Elkerton, J. "Software Interface Design," in G. Salvendy (Ed.) Handbook of Human Factors. 1987 NEW YORK: John Wiley & Sons.

Woods, D. D., "Visual Momentum: A Concept to Improve the Cognitive Coupling of Person and Computer," International Journal of Man-Machine Studies, Vol. 21, 1984, pp.229-244

Yadav, S. B., Bravoco, R. R., Chatfield, A, T., and T. M. Rajkumar, "Comparison of Analysis Techniques for Information Requirement Determination," Communications of the ACM, 1988, Vol. 31, No. 9, pp.1090-1097.

Yunten, T., and Hartson, H. R., "A Supervisory Methodology and Notation (SUPERMAN) for Human-computer System Development," in H. R. Hartson, (Ed.), Advances in Human-Computer Interaction, 1985, Norwood NJ: Ablex, pp.243-281.

Zdybel, F., Greenfield, N. R., Yonke, M. D., and J. Gibbons, "An Information Presentation.System," Proceedings of IJCAI "81, Vancouver, 1981, pp.979-984.

8. Appendix A-Interpretive Mapping

8.1 Constructing Information Displays

8.1.1 The Graphical-Interpretation Task

The process by which information is communicated from a computer to a user by graphical means is graphical visualization. Graphical visualizations are becoming increasingly prominent in state-of-the-art user interfaces, and this trend is certain to continue. There are two fundamental tasks in the design of any graphical visualization: the content-determination task is to decide what information to communicate graphically, and the graphical-interpretation task is to construct information displays to communicate that information. Previous sections of this report have addressed in detail the content-determination task and its many ramifications. In this Appendix we limit our attention to the possible role of an intelligent tool in the process of designing and constructing information displays for user interfaces. We recommend the adoption of three core principles in the design of any such tool. Previous approaches that conform to these principles have been restricted to a single subject domain and one or two types of information display. A uniform theoretical model that accounts for heterogeneous sets of subject domains and information-display types would have many advantages. The interpretive-mapping (IM) computational paradigm is a candidate for a uniform theoretical model. We describe the characteristics of this paradigm, and how it subsumes previous research. The IM paradigm can be used in the development of an intelligent tool for the design of information displays. It can also be used to automate the design and construction of these displays. A detailed example is provided showing how the IM paradigm may be applied to a subject domain and a type of information display for which there currently are no comprehensive intelligent design tools.

8.1.2 Problem Scope and Core Principles

The graphical-interpretation task is that of constructing information displays. Our analysis of the graphical-interpretation task is based on the following assumptions: the information to be communicated has been determined and cast into a suitable representation; appropriate information-display types have been selected; the content and sequencing of individual displays has been determined; and user-preference data is available. Attempting to aid the user in satisfying these assumptions is addressed in part elsewhere in this report, although there is still a tremendous need for further research on topics related to these assumptions. The graphical-interpretation task as we have defined it is still a difficult and complex problem for which there are no easy solutions. Each of the next three paragraphs describes a core principle that we believe must be followed by any intelligent tool for aiding or automating the construction of information displays.

Knowledge of the subject domain and the user's informational task is required for designing effective information displays. It is very difficult to design a good information display if nothing is known about the information that is to be communicated. Earlier work in this area assumed that knowledge of the subject domain and of the user's task was needed only for choosing the type of information display. The BHARAT system (Gnanamgari 1981) automatically selects appropriate chart-graphic techniques, e.g., bar charts and line graphs, based on the inherent attributes of quantitative tabular data: for example, fractional quantities that sum to one might be portrayed in a pie chart. Mackinlay's approach (Mackinlay 1986) designs superior chart-graph presentations automatically by taking a more knowledge-intensive approach to the problem, and by applying that knowledge in a design (as opposed to selection) process. It seems certain that this observation extends to other types of information display that communicate information from other sources.

All aspects of graphic design must be considered by an intelligent design tool. We state as a fundamental premise that information is communicated graphically through a full suite of topological, geometric, retinal (e.g., color, size, brightness), and symbolic structures (Bertin 1983). To ignore any of these visual structures, or the principles of graphic design that apply to them, will adversely effect the information displays that are produced. At best, superior designs that use the omitted structures will not be discovered; at worst, spurious visual structures that convey misleading information may be unknowingly introduced. A corollary of this principle is that an explicit association of semantic structures in the subject domain and visual structures in the graphical domain is necessary for any system to be able to reason effectively about the design and construction of information displays. Not only should a design aid consider all aspects of graphic design, but there must be some method for representing the association between the information that is being communicated and the visual structures that are being used to communicate it.

Accurate and efficient communication of information should be the primary criterion for evaluating visualization designs. Many lower-level criteria have been suggested to evaluate graphic designs. Tufte suggests measuring the ratio of ink to information communicated (Tufte 1983). Marcus describes a suite of criteria that is derived from the Swiss approach to graphic design: sans serif type styles, simplified imagery, open spaces, consistency of design, and use of a grid of implied lines for organizing and positioning graphical elements (Marcus 1980). For network diagrams the following criteria have been suggested: number of edge crossings (Batini et al. 1986; Rowe et al. 1987), diagram size (Batini 1986), degree of symmetry (Lipton et al. 1985), and degree of convexity (Chiba et al. 1985). Although adopting some or all of these criteria can sometimes lead to graphical presentations that communicate information accurately and efficiently, this is not always so. Worse still, these criteria often evolve from heuristics for good design to goals of the design process. Knowing when to ignore some of these lower-level criteria is essential to superior design. The higher-level criterion of accurate and efficient communication avoids this issue by definition, but it begs the question of how to measure accuracy and efficiency of information communication. Bertin suggests how this can be subjectively intuited from observation (Bertin 1983), but theory based evaluations is more desirable (Cleveland and McGill 1985, Kosslyn 1985). Even if such a criterion is difficult to measure effectively, that does not mean that it should be abandoned in favor of misleading lower-level criteria that can be measured.

8.1.3 Previous Work on Aiding and Automating Graphical Interpretation

The goal of automating the construction of interfaces to databases and knowledge bases provided the impetus for some of the earliest work in this area. The AIPS system (Zdybel et al. 1981) automatically chooses display formats from a fixed repertoire to display the contents of a knowledge base. The first large-scale implementation of a system that automated the graphical-interpretation task was Friedell's VIEW system (Friedell 1982). The automated synthesis of graphical-object descriptions for entities in a database is based on a knowledge of the entities in an annotated database, their relation to each other, and on the focus and expertise of the user. The layout of graphical objects in Information Spaces (the Apple Macintosh desk top is an example of a simple Information Space, or Ispace) is also performed automatically. Friedell extended this work to provide a general computational mechanism for automatically synthesizing graphical-object descriptions from primitive graphical elements (Friedell 1984).

Previous work on the automatic visualization of quantitative and relational data has focused on the selection and design of effective charts and graphs. The BHARAT system (Gnanamgari 1981) uses a decision tree to automatically choose an appropriate charting technique, based on the inherent characteristics of quantitative tabular data. This system does not conform to the core principles we have outlined since it ignores many graphic-design issues. The choice of charting techniques is made from a small, fixed set of human-designed charting techniques. This is a severe restriction, since there is a tremendous variety of techniques for the graphical display of quantitative information (MacGregor 1979; Schmid and Schmid 1979; Bertin 1983; Tufte 1983; Cleveland and McGill 1985).

Mackinlay (Mackinlay 1986) has taken a more sophisticated approach to automating the design of effective charts. The goal of his APT system is "to synthesize a graphical design that expresses a set of relations and their structural properties effectively" for the domain of relational data. The exact information communicated by the different elements of a chart display is encoded by means of logical predicates. This encoding defines a mapping from the subject domain of relational data to the graphical domain of chart displays (this resonates closely with the interpretive-mapping paradigm we introduce below). Design variation is achieved by merging the elements of a design that encode similar information. The APT system elegantly achieves its goal of automatically generating effective designs for charts, although the repertoire of chart-display elements is limited.

Feiner's APEX system (Feiner 1985) produces pictorial explanations of objects and actions in a 3D world. Graphical sequences are automatically generated to explain tasks in the 3D world, such as repairing a machine. Only initial experimentation of an ad hoc nature has so far been reported for the APEX system. Earlier related research in this vein concerned the automatic creation of simplistic computer animation (Kahn 1979) from story descriptions.

8.1.4 A Uniform Theoretical Model: The Interpretive-Mapping Paradigm

A model of previous research is shown in Figure 8.1. Subject data are represented in Entity-Relation (ER) data models. Each connecting line represents a different computational mechanism linking one subject domain with one or two information-display types in the graphical domain. This model has a number of disadvantages. There is considerable duplication of effort across the various computational mechanisms, yet the types of information display available for each subject is restricted. There is no prescriptive paradigm for future research. It is difficult to evaluate each distinct mechanism with respect to the core principles described above. It is not clear if a given computational mechanism can be used as the basis of both an intelligent design aid and an automated designer, or whether separate mechanisms are needed.

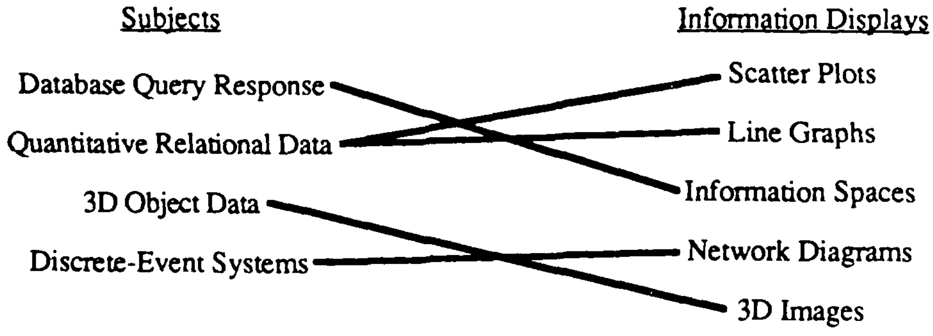


Figure 8.1 Previous research model.

For these reasons we propose the alternative research model illustrated in Figure 8.2. This model embodies a single computational paradigm for the graphical-interpretation task. A process of semantic abstraction is used to create semantic structures from the ER data models that describe a subject domain. The subject schema describes the semantic structures that are used in the IM paradigm. Semantic structures concern entities and their attributes. Entities correspond to objects and concepts in the subject domain. Entities have attributes associated with them. The classes of semantic structure that are used are binary relations over entities, partitions of entity sets, and attribute-value sets. The visual schema describes a parallel set of visual structures. Visual structures concern graphical marks and symbols. A mark or symbol also has attributes associated with it. There are three similar classes of visual structure: binary relations, partitions, and attribute-value sets. Examples of visual and semantic structures are shown in Figure 8.3.

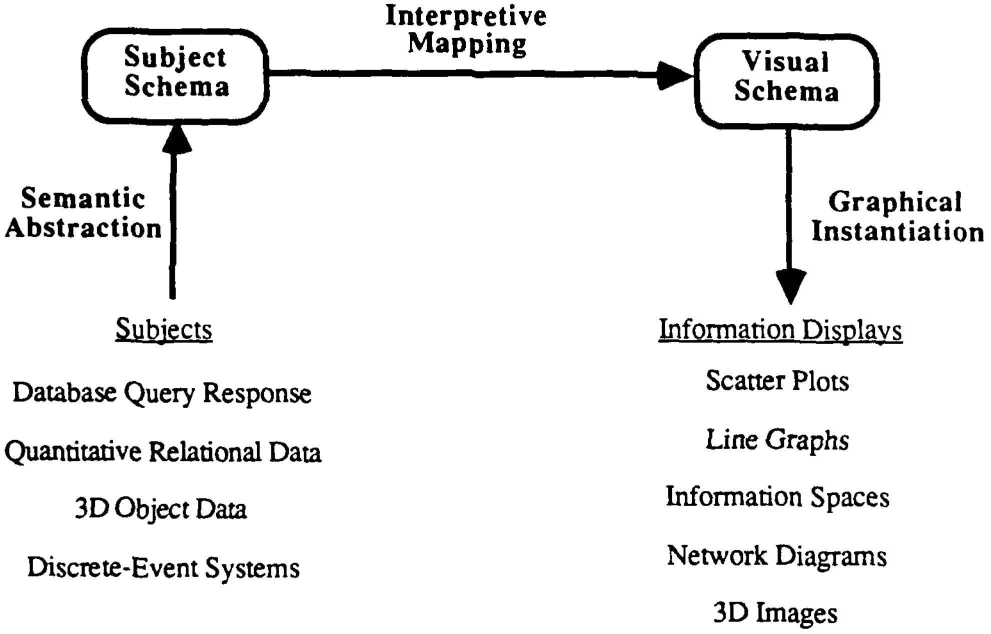


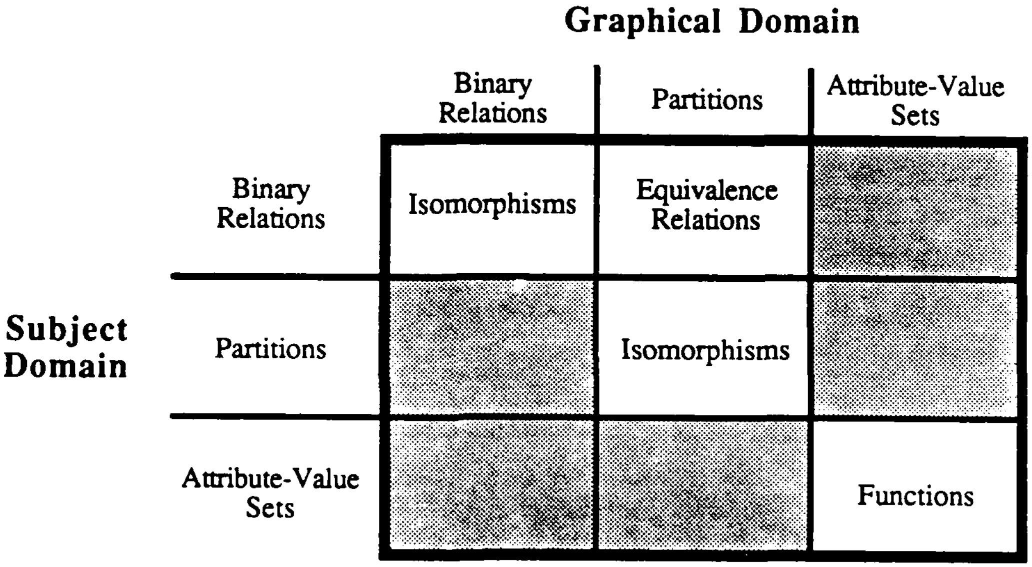
Figure 8.2 Current IM research model.

An interpretive mapping (IM) is a set of mappings from semantic structures in the subject domain to visual structures in the graphical domain. An element of an IM is called a structure mapping, of which there are three types: isomorphisms, equivalence relations, and functions. Each structure mapping identifies a particular visual structure that will be used to communicate a particular semantic structure. Sample structure mappings are shown in Figure 8.3. IM selection is the process of determining which structure mappings are included in an IM. Selecting an IM is a knowledge-intensive task requiring the following information: a description of available media

<u>Subject-Schema Concept</u>	<u>Example</u>
entity	**"Aircraft" is an entity with attributes "fuel" and "readiness."**
SEMANTIC STRUCTURES	
binary relation	$R = \{(a, b)$: aircraft a is stationed at base b$\}$
partition	$P = \{\{a: a.fuel < 10\%\}, \{b: b.fuel \geq 10\%\}\}$ — the "aircraft" entities are partitioned into those low on fuel, and those with adequate fuel.
attribute-value set	S = {ready, not_ready} is the set of values for attribute "readiness" of entity "aircraft."
<u>Visual-Schema Concept</u>	<u>Example</u>
mark, symbol	The aircraft symbol has attributes "color," "x," and "y," the latter two to indicate location in the information display.
VISUAL STRUCTURES	
binary relation	$R' = \{(a', b')$: the extent of symbol a' lies within the extent of symbol b'$\}$
partition	$P' = \{\{a': a'.x = 1\}, \{b': b'.x = 3\}\}$ — the symbols are partitioned into those aligned on line $x = 1$ and those aligned on the line $x = 3$.
attribute-value set	S' = {red, green} is the set of values for attribute "color" of aircraft symbols.
<u>Structure-Mapping Classes</u>	<u>Example</u>
ISOMORPHISM	
binary relation -> binary relation	R -> R': if aircraft a is stationed at base b, then the aircraft symbol a' is enclosed in the base symbol b'.
partition -> partition	P -> P': if $a.fuel < 10\%$, then $a'.x = 1$; if $b.fuel \geq 10\%$, then $b'.x = 3$.
EQUIVALENCE RELATION	
binary relation -> partition	$E(b) = \{\{a:(a, b) \in R$ for base b$\}, \{a:(a, b) \notin R$ for base b$\}\}$ is a set of equivalence classes induced from binary relation R. E(b) -> P': if aircraft a is stationed at base b, then $a'.x = 1$, else $a'.x = 3$.
FUNCTION	
attribute-value set -> attribute-value set	S -> S': if a.readiness = ready, then a'.color = green; if b.readiness = not_ready, then b'.color = red.

Figure 8.3 An illustration of the IM paradigm, with a hypothetical Air Force database as the subject domain, and information spaces as the type of information display.

capabilities; a description of user preferences; previous and future graphical-presentation tasks in the current dialogue; and extensive expertise in graphic design and human factors. An interpretive mapping can be thought of as a set of entries in an interpretive matrix : the rows correspond to semantic structures, and the columns to visual structures (see Figure 8.4). The interpretive matrix expresses the totality of ways in which information (in the form of semantic structures) can be communicated graphically (in the form of visual structures). Not all entries in the interpretive matrix correspond to a valid structure mapping, since a given semantic structure cannot be communicated by every visual structure.



		Graphical Domain		
		Binary Relations	Partitions	Attribute-Value Sets
Subject Domain	Binary Relations	Isomorphisms	Equivalence Relations	▨
	Partitions	▨	Isomorphisms	▨
	Attribute-Value Sets	▨	▨	Functions

Figure 8.4 The general form of an interpretive matrix.

An intelligent design aid needs to generate at least three kinds of advice. The first kind is advice relating to the selection of an IM. For example, should color or size be used to denote aircraft readiness? The second kind concerns the actual construction of an information display that conforms to the selected IM. Given that color has been selected to denote readiness, how should different levels of readiness be mapped into different colors? The third kind of advice concerns aesthetic and perceptual-limit considerations. What colors should be used to make semantic differences apparent and the displays visually pleasing?

For some applications it may be more appropriate to automate the construction of information displays. A process of graphical instantiation is then needed to automatically produce an information display that conforms to a given set of visual structures and that satisfies aesthetic and perceptual-limit standards. This process can be formulated as a constraint-satisfaction task, where conformity to a set of visual structures is formulated as a set of constraints, with additional constraints included to meet aesthetic and perceptual-limit criteria.

The IM computational paradigm has several advantages. It is capable of satisfying all ot the core principles we described earlier; it is a comprehensive paradigm: it provides a pattern for future research efforts as well as a way of conceptualizing previous work; and it can be used in the development of tools for both aiding and automating the construction of information displays. In the next section, we describe a current project that is following the paradigm to develop an automated designer for diagrams of network-flow systems, and we show how the IM paradigm is related to previous research.

8.2 A Discussion of Interpretive Mapping

8.2.1 Discrete-Event Systems and Network Diagrams

In this section we examine in detail how the IM paradigm can be applied to both a new subject domain and a new type of information display. The subject domain is that of discrete-event systems (DESs). A queueing system is an example of a DES. DESs are typical of the class of network-flow systems for which network diagrams constitute an important type of information display. Figure 8.5 contains an example of a network diagram that illustrates a queueing-system model of the disk subsystem of a computer.

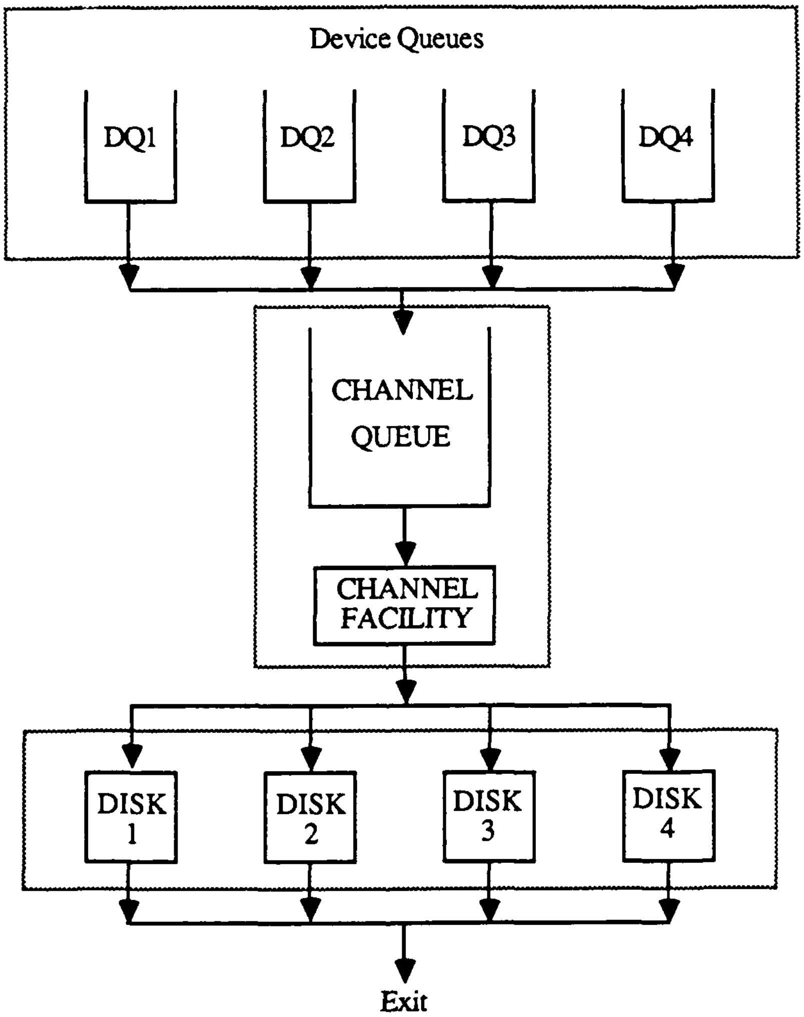


Figure 8.5 A network diagram of a disk subsystem in a computer (Herr 1986).

Before discussing how the IM paradigm is applied to the automated construction of network diagrams, we will first examine the human-designed diagrams in Figures 8.5 and 8.6. These apparently simple diagrams illustrate some interesting aspects of graphic design. For example, the visual structures of proximity, alignment, extent inclusion, and symmetry are determined by layout. A skillful graphic designer will use these visual structures to denote semantic structures involving queueing-system components and behavioral elements. Semantic structures can also be communicated by varying the shapes, sizes, orientations, colors, textures, label positions, and label fonts of icons and interconnecting edges. It is instructive to examine both diagrams with the purpose of identifying the differences in visual structure between them, and to consider the associated differences in semantic structure.

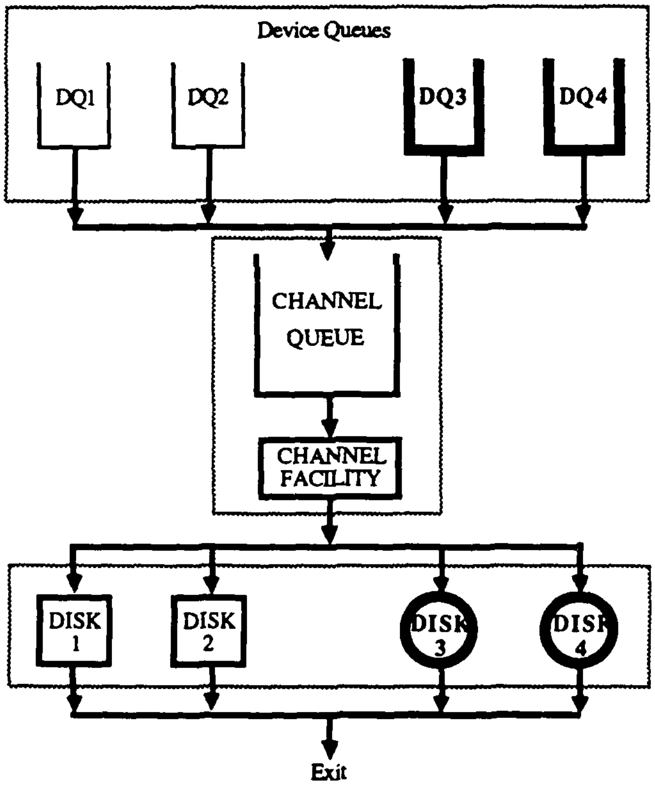


Figure 8.6 A network diagram for a differently-configured disk subsystem.

8.2.2 An ER Model for Discrete-Event Systems

Below we consider more completely the subject domain of queueing systems and the types of informational tasks that users will want to perform with network diagrams. The ER model we have chosen to represent the structure and function of DESs is based on the set of modeling primitives used by simulation researchers (Banks 1984). This traditional model is augmented to express subsystem-hierarchy relationships between entities, subprocess-hierarchy relationships between activities, and specialized classes of entities and activities. These additional features of the model would not be relevant for simulation and analysis of DESs, but they are necessary for supporting the semantic structures needed to graphically communicate the structure and behavior of a DES. An overview of the ER model is given below:

- An entity can be a server, a queue, a job, or a system. Each entity can have named numerical attributes associated with it. Entities of the same genus can be separated into specialized classes, e.g., servers in a banking system might be divided into human tellers and ATMs.

- An activity can be an input, output, service, transfer, or process activity. Input, output, and service activities are all associated with a single entity. Each transfer activity is associated with a pair of entities. System connectivity is indicated by the transfer activities between entities. Activities can also have named numerical attributes.

- The subsystem hierarchy indicates which entities belong to which subsystems. The process hierarchy indicates which activities belong to which process activities.

Certain information about a queueing system will always be communicated by a network diagram, such as connectivity information, and the general function of its parts. Other information should be supplied only when needed for the user's task. The ER model must also permit the representation of this meta-information, which might include the following features:

Selective Emphasis

Certain entities and activities in the system that share some common bond should be visually emphasized to facilitate quick identification, e.g., by sharing a common color.

Absolute Attribute Quantification

Numerical attributes associated with entities and activities in the system should be accurately communicated, e.g., through the use of labels, or proportional icon size.

Comparative Attribute Quantification — The relative ordering of numerical attribute values should be indicated, but the actual values may be indicated only approximately, e.g., though the use of icon value (brightness) or texture density.

System Structure — System structure based on subsystem hierarchy or process hierarchy should be illustrated, e.g., by proximity or overlap of icons.

Generalization/Specialization — Association and differentiation between entities and activities of the same genus should be indicated, e.g., by sharing a common icon symbol or color.

System Hierarchy Level — A certain level of the hierarchical system description should be emphasized, e.g., by imposing topological or geometric constraints among entities at only the selected level in the hierarchy.

8.2.3 Semantic Abstraction, IM Selection, and Graphical Instantiation

The foregoing descriptions indicate the nature and scope of information that is needed to construct the semantic and visual structures for this application of the IM paradigm. Next we look at a small selection of semantic and visual structures and walk through the algorithmic tasks that are necessary to automatically construct an information display.

Figure 8.7 shows a sparsely-populated interpretive matrix, with semantic and visual structures that are appropriate for the subject domain of DESs and the graphical domain of network diagrams. It is useful to relate this interpretive matrix to the general form of an interpretive matrix shown in Figure 8.4. The semantic structures are self-explanatory, but it is interesting to think about how these structures might be constructed from the information described earlier in this section. That task is, in fact, the algorithmic task of semantic abstraction.

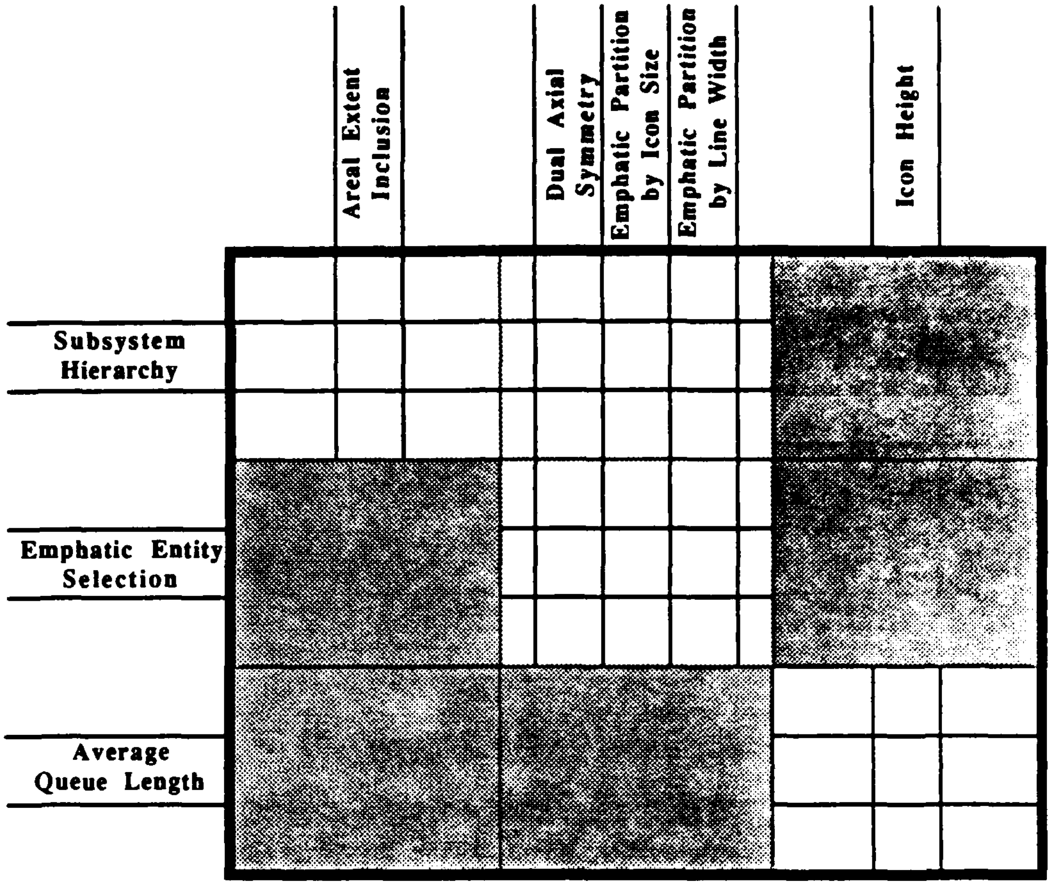


Figure 8.7 Selected rows and columns from an interpretive matrix.

The second algorithmic task is to select the structure mappings for an IM. Figure 8.8 illustrates a partial selection that is at an impasse. The subsystem-hierarchy semantic structure will be communicated by the visual structures of areal extent inclusion and perceptual grouping due to dual axial symmetry. Emphatic entity selection will be communicated by sizing icons appropriately. But then the only visual structure left with which to communicate average queue length is the icon height. Unfortunately, this visual structure interferes with partitioning icons by size, a visual structure to which we have already committed. This impasse indicates the knowledge and expertise that is required in the IM selection task.

A solution to the problem is to use a partition by line width to communicate emphatic entity selection, thus allowing a valid structure mapping from average queue length to icon height to be included in the IM. The complete IM is shown in figure 8.9.

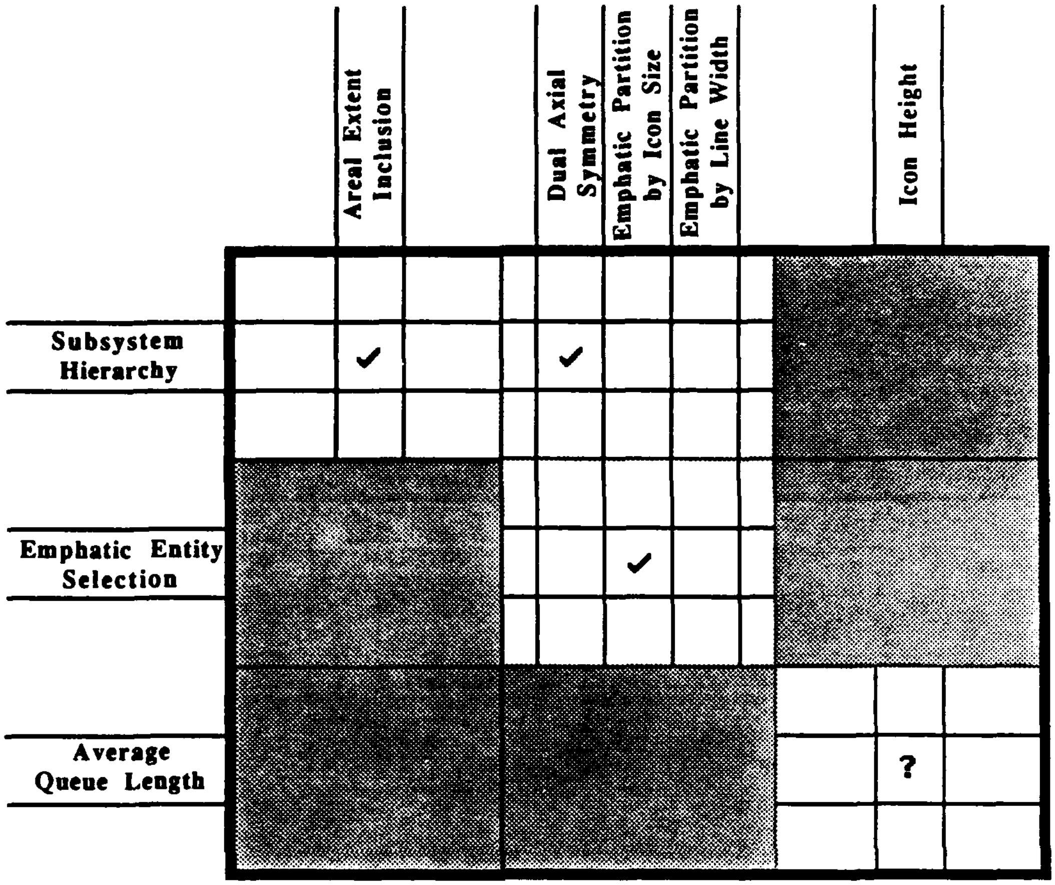


Figure 8.8 An impasse in the selection of an IM.

Given an IM, the remaining algorithmic task is to graphically instantiate a network diagram which conforms to the visual structures selected in the IM. Such a diagram is drawn in figure 8.10. A quick inspection will confirm that the diagram does conform to the visual structures chosen in Figure 8.9. Two of the three semantic structures are communicated quite effectively. The diagram, however, has one significant flaw. The selection of icon height as a means of communicating average queue length would have been much better if the device queue icons had been aligned by their bottoms rather than by their centers. But their centers were aligned due to the use of the dual-axial-symmetry visual structure. This is a much more subtle interaction between structure mappings than we saw in Figure 8.8, but it should also have been noticed during the selection of the IM. This indicates once again the crucial role of knowledge and expertise in the IM selection task.

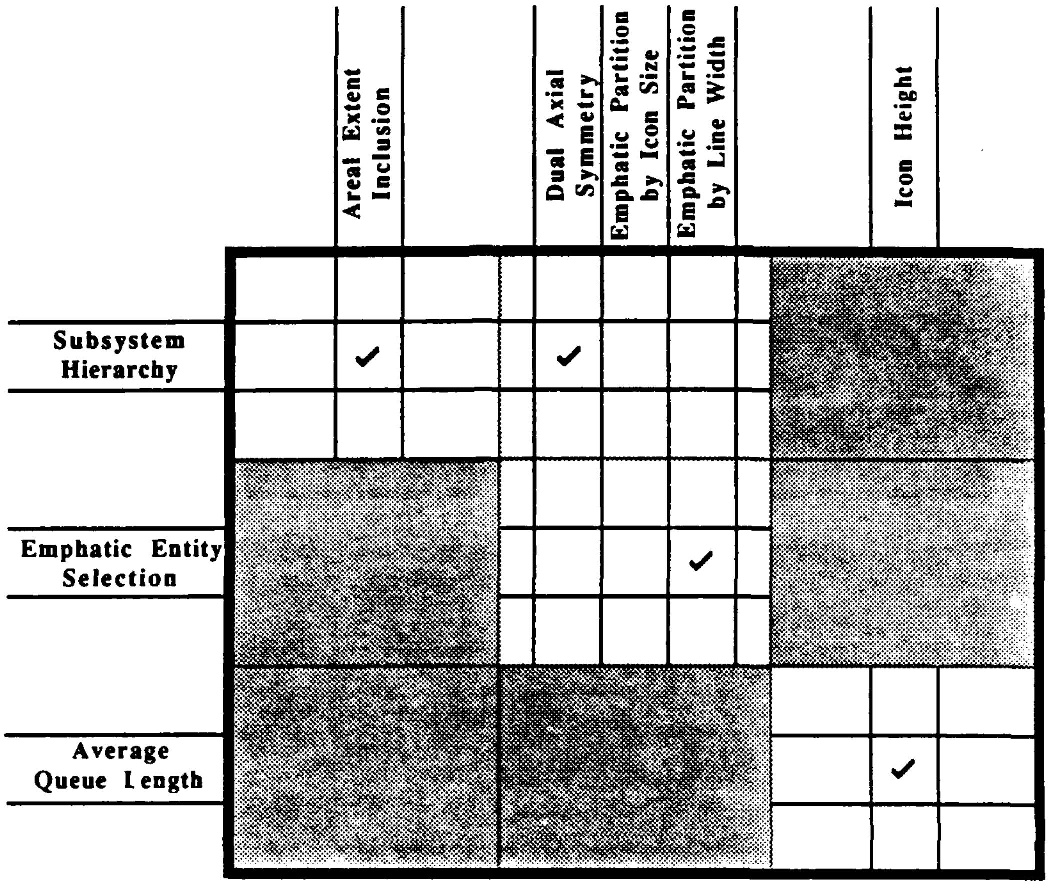


Figure 8.9 A completed IM.

8.2.4 Using the IM Paradigm to Generate Advice

Rather than automatically constructing the network diagram in Figure 8.10, the goal of an intelligent design aid would be to aid a person in designing a diagram, by providing the three kinds of advice and criticism we identified in Section 8.1.4: 1) selecting an IM; 2) constructing an information display that conforms to an IM; and 3) applying aesthetic and perceptual-limit considerations. In this context, IM selection now becomes a task that is, perhaps, performed jointly by the user and the computer, with the computer evaluating the user's choice of structure mappings, suggesting alternatives, presenting exemplar network diagrams that illustrate particular structure mappings, and identifying contradictions and side-effects within a nascent IM. Once an IM is selected, the design adviser might perform the graphical-instantiation task one or more times to automatically generate a sele :tion of plausible network diagrams. A design adviser might also act as a critic, evaluating a user-instantiated network diagram with respect to the selected IM.

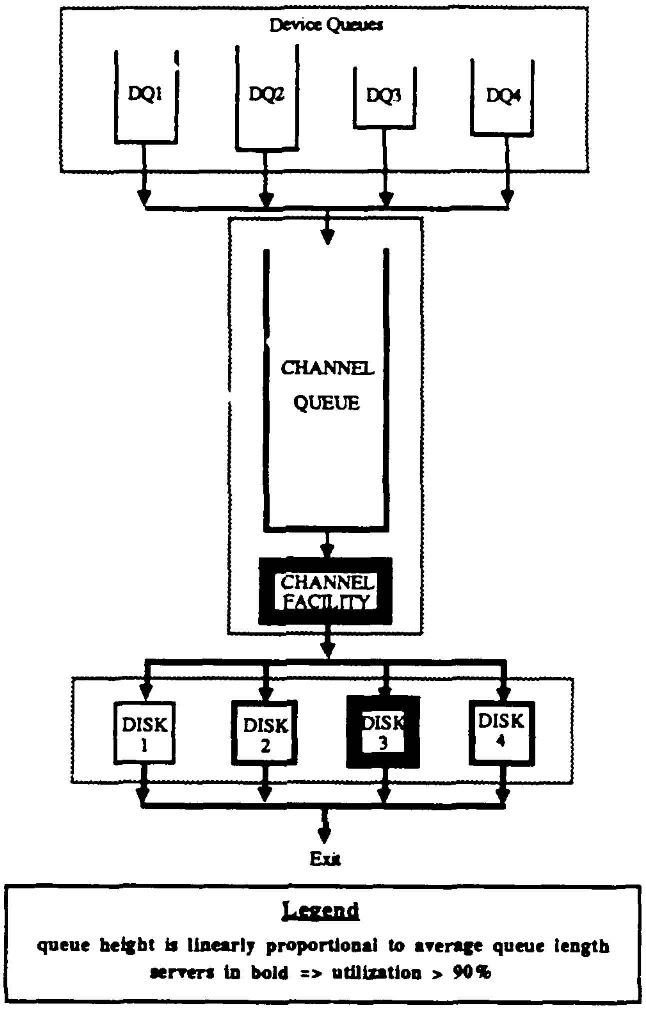

Device Queues
DQ1
DQ2
DQ3
DQ4
CHANNEL
QUEUE
CHANNEL
FACILITY
DISK
1
DISK
2
DISK
3
DISK
4
Exit
Legend
queue height is linearly proportional to average queue length
servers in bold => utilization > 90%

Figure 8.10 A variation of the network diagram from Figure 8.5 that conforms to the visual structures in the IM of Figure 8.9.

8.3 Conclusions

In Appendix A we have described a theoretical approach to the task of graphical interpretation, the IM computational paradigm. This paradigm provides a way to think of previous research, and outlines a technical approach for future research. We have discussed how the IM paradigm can be used in the contexts of both an intelligent design aid and a fully-automated designer. Current research is directed at using the IM paradigm to automate the design of network diagrams depicting DESs.

The IM research conducted to date suggests the pursuit of three research goals. One goal is additional research into the use of the IM paradigm in the context of intelligent design advising. The second research goal concerns the practicality of system implementation. The IM paradigm provides a uniform theoretical model for graphical interpretation, but it is not yet clear how advantageous this is in terms of combining and merging aspects of system implementation across different applications. For example, is it possible for an intelligent design adviser to have a common module for considering design issues related to color for both network diagrams and chart graphics? The second research goal thus concerns the identification of common and specialized tasks in the IM paradigm for different types of information display. The third research goal arises from the observation that information displays do not usually occur in isolation, but as part of a man-machine dialogue. This brings up issues related to the division of information across sequences of displays, and consistency in the use of visual structures within a dialogue. A research effort directed at understanding the impact of these dialogue issues on the design and construction of information displays is needed.

9. Appendix B-System Reviews

9.1 Arens et al. 1988

Arens, Y., Miller, L., Shapiro, S. C., and Sondheimer, N. K., "Automatic Construction of User-Interface Displays," paper presented at The 1988 Conference of the American Association of Artificial Intelligence, St. Paul, Minnesota, August 22-26, 1988.

Description

Target User

Integrated Interfaces is an Intelligent Interface that supports the integration of display modes, dynamically produces multi-modal displays, and supports generalization and enhancement. This system automatically generates dynamically produced displays based on explicit rules of information presentation.

Decision-Aiding Style

The style of the Integrated Interfaces system is based on the premise that the most difficult interfaces to build are those that dynamically create displays. The goal of the system is for it to be able to automatically choose between multiple media, multiple modes, and multiple methods. The designers of the system believe that simply having several modes available is not enough, their use must be integrated. Different items of information must be distributed to appropriate modes, the amount of redundancy should be limited to the amount needed to establish co-reference, and the different presentations modes must all work from a common meaningful representation to ensure accurate presentations. Furthermore, the interface system integrating a set of modes must be capable of dynamically producing different displays. Fixed multi-modal capabilities are not sufficient in a rapidly changing environment. Techniques must be employed to support generalization and enhancement of displays over time since the initial interface is certain to require upgrading.

Architecture

The architecture of the Integrated Interfaces system relies heavily on existing knowledge and rule based AI technology. NIKL is used to model the entities of the application domain and the facilities of the user interface. Rules connect the two models. These rules range from application specific to general rules of presentation. The situation to be displayed is asserted into a PENNI database. A Presentation Designer module interprets this data using the domain model, chooses the appropriate rules to use in creating the display, and creates a description of the desired display in terms of the interface model. Device drivers interpret such descriptions to create the actual displays.

Integrated Interfaces is written in Common Lisp and executes in the X window environment under UNIX on HP 9000 Model 350 workstations. Displays are presented on a Renaissance color graphics monitor. The map graphic modality is support by ISI's Graphics Display agent. Menus and forms are created using QFORMS. Natural language output is produced by ISI's Penman system.

Design Heuristics

Presentation design is described as the task of realizing the application domain categories within which a request for display information falls, selecting the appropriate rules that apply to those categories, and redescribing the application in terms of the request into appropriate display terms. Realization relates the facts about instances to the abstract categories of the model. Selection works by allowing and ensuring that appropriate mapping rules are to be chosen, allowing for additivity. Redescription applies the rules, mapping each aspect of a common-sense view of a presentation into an equivalent presentation form.

Evaluation Metrics

The power of a model-driven presentation design, such as the one employed by Integrated Interfaces, is in its flexibility. The designer of a system does not specify rigidly in advance in what form information will be requested from the user, and how data and results will be displayed. Instead, models contain descriptions of the types of information the application deals with, and the types of graphical tools and instruments available. The rules for presentation enable the system to generate, on demand, displays deemed appropriate for given user needs.

Evaluation

Strengths

1. Integrated Interfaces attempts to automate the graphical interface design process.
2. The Integrated Interfaces system is capable of dynamically creating menus for choosing among alternate actions, and more complicated forms for specifying desired information.
3. Because it uses existing technology, Integrated Interfaces is described by its designers as a cost efficient system.

Weakness

1. Integrated Interfaces cannot tailor its presentations to individual user preferences.

2. The Presentation Designer, within the Integrated Interfaces system, is not aware of the purpose of the display and, thus, cannot tailor the display to specific users' requests.
3. Integrated Interfaces' internal description of the display is not rich enough to allow a user to alternate between references to screen entities and their denotations.

Problem Coverage

Integrated Interfaces attempts to addresses the problem of automatically constructing integrated user-interface displays. This system employs the use of system models and rules to present retrieved information using a combination of output modes - natural language, text, maps, tables, menus, and forms. It can also handle input through several modes - menus, forms, and pointing.

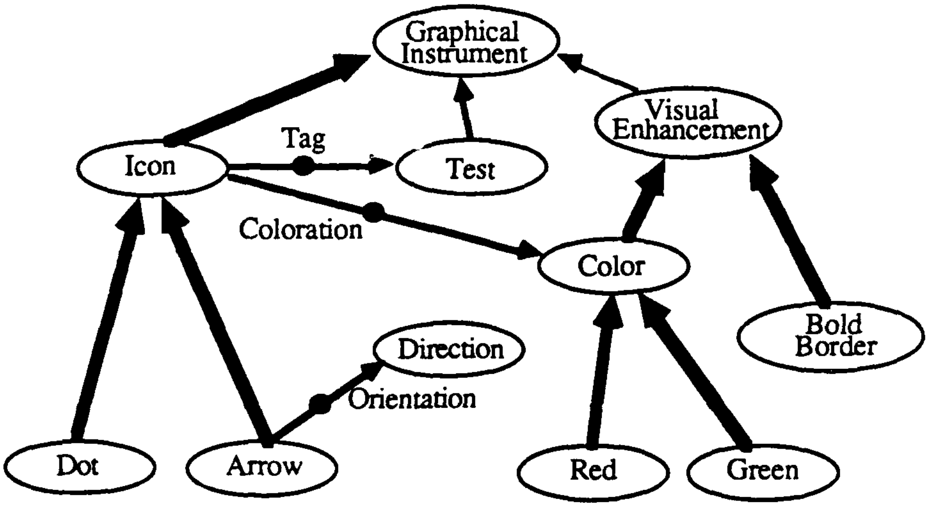


Figure 9.1 Fragment of the Integrated Interface conceptual model

The application model identifies the categories of objects and actions in the application's view of the world. Subclasses of relations are present among categories, as well as relationships between objects and actions. An application model needs to be created for each new application interface.

The interface model describes the categories of objects and actions of the interface world. The objects include windows, tables, maps, text strings, and icons. The actions include creation, deletion, movement, and structuring of displays. Describing all interface modes together in a single model is a necessary basis of integration.

The rules map objects from the application model into objects in the interface model. Rules take their conditions from the application model and when used in conjunction with the interface model, allow integration. They can be used to distribute information among modes, minimize redundancy, and coordinate presentations. Integrated Interfaces allows both low-level application specific rules and high-level application-dependent rules.

Current Applications

Integrated Interfaces has been used in applications that model the domain of Navy ships operating in the Pacific Ocean.

9.2 Beach and Stone 1983

Beach, R. and M. Stone, "Graphical Style: Toward High Quality Illustrations," Computer Graphics, 1983, Vol. 17, No. 3, pp.127-135.

Description

Target User

TiogaArtwork was designed to coordinate with the Tioga document preparation system and an interactive illustration program called Griffin. This system was designed to provide the document designer with a tool that can be used to ensuring that related illustrations have a consistent appearance within a technical document.

Decision-Aiding Style

Traditionally, a graphic designer uses a style sheet to communicate how to render a document or image to a compositor. The purpose of the style sheet is to ensure consistency and design discipline within a document, and to provide a rapid and effective means for specifying that discipline. The TiogaArtwork system embodies the concept of graphical style by presenting a representation that explicitly specifies the stylistic properties of an illustration.

Architecture

The TiogaArtwork system was developed in the Cedar programming environment at Xerox PARC. Cedar is considered both a language, derived from Mesa, and a computing environment. TiogaArtwork executes on a Dorado processor with a 1024 by 808 pixel monochrome display and, optionally, a 640 by 480 pixel color display.

The architecture of the TiogaArtwork system is based on features of the Tioga document preparation system. The Tioga system implementation consists of an interactive text editor and a batch-oriented typesetter. Documents in Tioga consist of two files: a file of hierarchically structured text and a file of formatting style rules. A document in the TiogaArtwork system is a tree of nodes arranged in a hierarchical structure. Some of the branches of the tree represent paragraphs and some represent illustrations. In TiogaArtwork, illustrations are text instructions for graphical representation.

TiogaArtwork is the annexed part of the system that interprets hierarchically structured graphics and style rules. TiogaArtwork translates the representation of an illustration into a set of calls on the Cedar graphics package. The Cedar graphics package implements a full set of geometrical transformations and clipping operations. Shapes are described as a set of analytical outlines that are filled with either a flat color or an image.

Design Heuristics

When using TiogaArtwork, it is considered that graphic arts quality and design issues are sensitive to the graphics designer's cognitive understanding of the document's style requirements Often the graphics designer asks "Why can't the program do this?" while offering suggestions as to the appearance of the final result.

Evaluation Metrics

Evaluation of an illustration produced on the TiogaArtwork system is done by the designer/user.

Evaluation

Strengths

1. The TiogaArtwork system is a flexible tool for experimenting with different ways of representing illustrations in a document.
2. TiogaArtwork uses a graphical style paradigm for presenting illustrations.

Weakness

1. The TiogaArtwork system can only use geometric information in the document body and puts all rendering parameters in the style rules.
2. TiogaArtwork has no empirical schema for interface design heuristics or evaluation metrics.
3. There is no facility to import illustrations into TiogaArtwork from other applications.
4. TiogaArtwork has no dynamic reconfiguration capability for layout.

Problem Coverage

The TiogaArtwork system controls the stylistic aspects of illustrations by using a set of explicit style rules to define the design guidelines. These style rules control the uniform "look" of a set of illustrations to reflect different publishing styles or different media.

The procedure for generating an illustration is to make a draft using Griffin and then convert the illustration to the special Tioga document. Once the illustration is in document form, it is possible to adjust both the style and the content using a combination of Tioga and

TiogaArtwork. The illustrations can be previewed on a display screen using the Cedar graphics package or converted for printing.

The combination of using the Tioga document structure and representing the graphics as text allows TiogaArtwork to interact easily with the Tioga editor and the Tioga typesetter. This is advantageous in a number of ways. One advantage is the ease of creating and editing figures. The Tioga editor does not react to any graphics property, so the text and style properties for graphics can be edited in a normal manner. This means that no special editor was created to manipulate the graphics, but instead, a conversion routine was written that translated Griffin format to Tioga format and automatically generates style rule properties from the Griffin style attributes. Another advantage of using TiogaArtwork is that it permits any text to be type set in any illustration.

Current Applications

The TiogaArtwork system has been used to produce publication quality documents.

9.3 Borning and Duisberg 1986

Borning, A., and R. Duisberg, "Constraint-Based Tools for Building User Interfaces," ACM Transactions on Graphics, 1986, Vol. 5, No. 4, pp.345-374.

Description

Target User

Constraint-based tools are useful in constructing user interfaces for interface designers, and creating flexible environments for nonprogrammers.

Decision-Aiding Style

Constraints allow a declarative style of user interface specification. With constraints, the user specifies what relations are to hold. It is then the responsibility of the underlying system to decide how to maintain those relations.

Since flexible environments for nonprogrammers are desirable, constraint-based tools allow users to avoid programming when possible and instead support the construction of objects using concrete, interactive, graphical techniques. The user is provided with a kit of useful building blocks which can be put together in flexible ways to assemble a given interface.

Architecture

A constraint is a relation that must be maintained. Maintaining that relation is left up to the underlying system, instead of the designer. A constraint contains both a declarative description of the relation and a set of procedures for making that relation hold, which are used by the underlying constraint satisfier.

A static constraint describes a relation that must hold at all times. Each such constraint is represented as a predicate and a set of methods that can be invoked to satisfy the constraint, reflecting its dual nature as both a description and a command. The predicate may be used to test whether or not the constraint is satisfied. The methods are procedures that are alternate ways of satisfying the constraint. If any method is invoked, the constraint will be satisfied.

Temporal constraints have a somewhat different representation. Two sorts of temporal constraints exist - continuous which are time differential constraints and time function constraints, and discrete which are trigger constraints. Time differential constraints support animation for describing the behavior of a component. Time function constraints are used for expressing the dependence of a value on an explicit function of time. Trigger constraints are particularly convenient for specifying how images are to move and change in response to particular events of the underlying executing process.

To satisfy static constraints, the constraint satisfier looks for constraint parts whose state will be completely known at run time. If such a part is found, the constraint satisfier will look for one-step deductions that allow the states of other parts to be known at run time, and so on, recursively. When propagating known states, the constraint satisfier can use information from different levels in the part-whole hierarchy. If the state of an object is known, the states of all its parts are known. If the states of all its parts are known, the state of an object is known.

Temporal constraints satisfaction is similar in that it involves a planning stage and results in the invoking of methods at run time. However, these run time methods do not affect values directly but instead place events in a queue to occur in the future. The responses to the triggering of temporal constraints may be quite involved, entailing sequences of events or scripts, or starting up a procedure.

Design Heuristics

Constraints are used in a number of ways to design user interfaces. They are used to maintain consistency between the underlying data and its graphical depiction on the screen, to maintain consistency among multiple views of the data, to specify how information is to be formatted on the display, to specify animation events that are to occur when a given event occurs in the underlying system, and to specify attributes of objects in an animation.

Evaluation Metrics

The individual building blocks of an interface have constraints the specify aspects of their behavior. The constraint satisfaction process ensures that, when the building blocks are assembled to make some interface object, all included constraints will be satisfied.

Evaluation

Strengths

1. Constraint-based toolkits provide a research facility for graphically defining new kinds of constraints.
2. Interfaces are built from objects with graphical representations.
3. Constraint-based tools allow a graphical interactive declarative style of interface design.

Weakness

1. No empirical schema for interface evaluation.

2. In applications in which constraint changes are made frequently, constraint satisfaction takes a noticeable amount of time.

Problem Coverage

The constraint-based tool kit is designed to handle a number of problems:

- Maintenance of consistency between data and displayed information is a common problem in user-interface design. An advantage of constraints as a way of handling this problem is that a constraint relation can satisfy itself bidirectionally.

- When several pieces of information are to be displayed, the items must be arranged somehow on the screen. Some aspects of the layout are left up to the user. In other cases at least some decision about the placement are left up to the system. In the latter case constraints provide a natural way of stating and implementing layout requirements.

- Animations fall into two categories: animations of continuous physical phenomena and animations of algorithms. In the continuous case, the constraints on behavior take the form of differential equations of motion for the system components. With animations of algorithms, an interactive graphical interface to an underlying running program is provided.

- Two views of the system are employed: a use view and a construction view. The use view shows an object's normal appearance, as it will appear whenever is it used as a part in some larger object. The construction view contains additional information to give the object its desired behavior. The user can build up the use and construction views using the same graphical tools that are available for construction any other object. The system will synthesize constraints accordingly.

Current Applications

ThingLab is a constraint-based tool kit for building things such as geometric demonstrations, simulations of physics experiments, user-interface frameworks, dynamic documents, and graphical calculators.

Animus extends ThingLab by introducing temporal constraints that allow the implementation of algorithm animations and autonomous simulations.

9.4 Corbett 1986

Corbett, J.D., Chips: A Toolkit for Editing Interfaces Programmer's Guide, Learning Research and Development Center, September 15, 1986.

Description

Target User

The Chips Toolkit is a set of tools intended to allow users with a range of skills to engineer graphical interfaces. At the higher levels, Human Factor Engineers can interact with the predefined functionality of the design tool to produce interfaces. At lower levels, Computer Scientists can define and create new tools and specify system interactions. The levels are specified as follows:

1. Allows the user to manipulated screen objects and use existing functionality.
2. The user can define new actions in terms of predefined Chip functionality.
3. The users can define new types of functionality in terms of Chip tools.
4. Allows the user to access system interaction.

Decision-Aiding Style

Chips was designed to lower the threshold of modifiability of graphical interfaces. This means that a Chips interface is designed to be modified by doing straightforward adjustments like the direct manipulation of screen objects, changing parameters of objects, tex diting object source code, etc. This is consistent with the Chips designers' philosophy that "A good interface is almost never designed without extensive, empirical work with actual users. To make matters worse, user interfaces are typically very difficult to implement, requiring elaborate coordination of user actions, screen display, and underlying functionality. Interfaces must also be comprehensible and must deal with user errors gracefully. Consequently they tend to be complex and brittle. This situation makes changes demanded by empirical results very expensive."

Architecture

The Chips Toolkit is implemented in LOOPS and Interlisp-D and executes on a Xerox Dandelion Machine. Therefore, Chips graphical interfaces are used for applications that reside in the Loops/Interlisp-D environment.

The Chips architecture is woven in the roots of object-oriented programming. Its key aspects are organization by multiple inheritance and homogeneous names defined over heterogeneous types.

Design Heuristics

In designing a Chip's interface, the implementers must rely on their own memory and understanding of the requirements for the graphical interface at hand. The Chips designers consider this to be approachable through the development of an environment that enables developers of systems to think about those systems in the same terms as the users of the systems.

Evaluation Metrics

Evaluation of a Chip's created interface is done by prototyping and testing the interface on a small test group of potential users. The users' critiques of the interface are then used by the designers to refine the interface. This is repeated until the interface is considered suitable to both users and designers.

Evaluation

Strengths

1. Philosophy of Designing Interfaces using Chips: The Chips Toolkit was designed to build graphical interfaces that would benefit the novice user but not hinder the expert and to help elevate the novice to the level of the expert incrementally while using the interface (diSessa's Continuous Incremental Advantage).

2. Attempts to lower the threshold of modifiability of graphical interfaces through a direct manipulation metaphor (explained above).

3. Chip interfaces are built from objects with graphical representations.

4. Graphical artifacts are linked to program source code. "One can go directly to the functions and data responsible for any displayed object directly by pointing to that object."

Weakness

1. No empirical schema for interface design heuristics or evaluation metrics.
2. How far can the direct manipulation metaphor for the modifiability of interfaces be pushed until it becomes unusable?

Problem Coverage

Once the task has been defined for an application, the implementers of the application will (most likely) use an engineering interface to create, modify, and, otherwise, access different modules of the new system. All interaction to and from the system, no matter what the level of design, will be created using the Chips Toolkit. This step in the system's development is usually done by skilled programmers, since it involves regular Interlisp-D programming.

Once the system, itself, is exercised to meet the programmers' specification using the engineering interface, the programmers will work with a Human Factors Engineer to create the first version of the user's graphical interface. The Human Factors Engineer (HFE) has, in parallel with the system creation, defined the needed functionality of the user's interface. This programmer/HFE interaction will allow the programmers to create any new types of functionality in terms of Chips tools and then, effectively, drop out of the design phase. Now the HFE can work at a higher level of abstraction in the Chips graphical interface development.

When the first version of the user's interface is completed, the HFE tests the interface on a representative set of typical users of the application. The HFE observes how the system is being used and what modifications are needed. The users critique the system to the HFE. This HFE/user interaction allows the HFE to define new actions in term of the predefined Chips functionality. This is an iterative design process that may be repeated numerous times before the interface meets the HFE specifications.

In the final phase of the development, the user is allowed to manipulate screen objects and use the existing functionality of the interface to customize (within limits set by the HFE) itself. This permits the HFE to drop out of the design loop. The graphical interface is now considered complete.

Current Applications

Circuits -- A digital logic simulator
Avionics Tutor -- A flight simulator
Novice Visual Programming Language (NVPL)
Marble World -- A microworld environment for counting
HyperText
Personal Management Tool

9.5 Ferrante and Tench 1986

Ferrante, R. D. and K. A. Tench, "TIDAS: An Interactive Man-Machine Interface Prototyping Environment," Proceedings of CHI '86, pp.1478-1483.

Description

Target User

The Tactical Intelligent Decision Aids System (TIDAS) permits an interface designer to develop a display in an interactive, iterative fashion. TIDAS provides the designer flexibility in developing displays of his own choosing and in building and tailoring the accompanying dialogue.

Decision-Aiding Style

TIDAS is designed to combine the advantages of a rapid prototyping environment with those of system simulation to provide designers with a range of tools for display development. Such a coupling of design and simulation is viewed as essential if the displays are to be capable of providing a useful system interface during dynamically changing situations. This is because simulation of complex, dynamic events is considered the only reliable method of capturing time dependent behavior.

Architecture

The TIDAS system is implemented on a SYMBOLICS 3640 using Zetalisp, Flavors (an object oriented programming language) and a monochrome monitor.

The TIDAS system consists of a database, knowledge base, information processing routines, and man-machine interface integrated via planning/scheduling functions. The TIDAS functions are a display editor which supports the interactive definition of designer displays, and a simulator which supports the testing of prototype designer displays.

The TIDAS Display Editor allows for simplified interaction with the Symbolics window system. It was designed for display designers, and does not require a detailed knowledge of Lisp or Flavors. The display editor manages the pane naming conventions and constraint propagation required for use on the Symbolics window system. Also, it transparently provides the proportional layout and dependencies required by the window system. In this respect, the display editor is designed to provide users a graphic programming environment that requires a minimum of low-level programming.

The simulator was developed to allow for a realistic evaluations of display designs. The simulator is designed to emulate the movement and activities within the display.

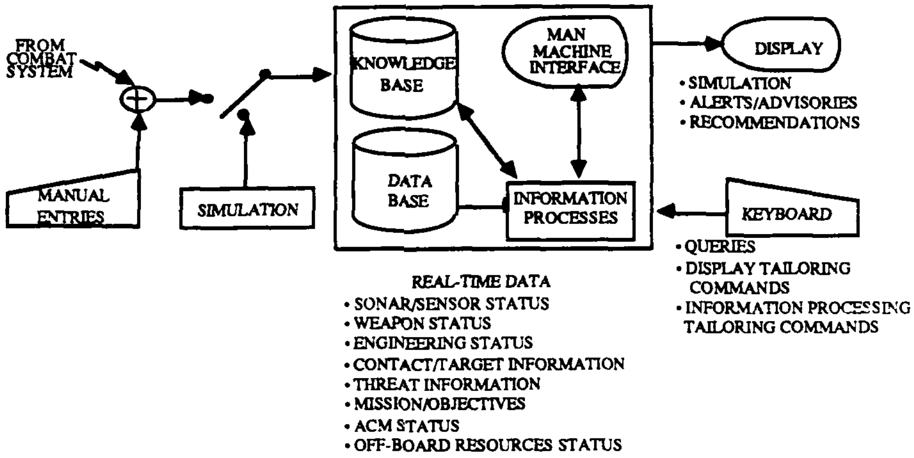


Figure 9.2 The TIDAS system concept.

Design Heuristics

Display layout is first interactively specified, then tested and modified as necessary to achieve the final display definition. This interactive approach to user interface design is functionally different from approaches based on formal specification. Formal specification of a user interface implies that all the interaction requirements are precisely known beforehand. In contrast, TIDAS does not assume complete a priori knowledge of the user's requirements. Rather, it allows the iterative refinement of display specifications. It is more akin to a system that provides an environment for rapid prototyping of the user interface. Therefore, it enables the design of fundamentally new displays, the requirements of which may be difficult or impossible to specify in advance.

Evaluation Metrics

TIDAS' simulator allows the designer nearly immediate feedback on the display implementation for evaluation. To fully evaluate the display, TIDAS must be used to alter the speed of the simulation, and change the type of events that cover the full field of expected situations. After the prototype has been developed, the TIDAS environment should be used to provide standard test conditions for a set of proposed display designs. Evaluation by end users can then be performed as part of the same process as display development.

Evaluation

Strengths

1. TIDAS is implemented as a design environment rather than as a design consultant or expert.
2. User interfaces can be created without a driving application system.
3. TIDAS combines design and simulation into a usable environment.

Weakness

1. No empirical schema for interface design heuristics or evaluation metrics.
2. With every additional feature, TIDAS increases with overall size and complexity.

Problem Coverage

Typically, the designer develops an initial display specification using the Display Editor and enters the simulator to get a "look and feel" for how the display will actually perform in use. The simulator can be used to replicate the dynamic aspects of the application environment as they impact the display.

During the simulation, the user may view the evolving situation and interact with the system via the controls which were included when the display was created or edited. In this manner, system engineers and human factor engineers can experiment with the utility or various layouts and command capabilities. If alterations are desired to the display layout, the user may re-enter the Display Editor and modify the definition of the interface.

Current Applications

TIDAS has been used as the man-machine interface construction tool in Anti-Submarine Warfare (ASW) systems.

9.6 Flanagan et al. 1987

Flanagan, D. P., Blue, D. V., Giacaglio, R. A., Lenorowitz, D. R., and E. C. Stanke, Rapid Intelligent Prototyping Laboratory (RIPL), Architecture and Use. Computer Technology Associates, January 1987.

Description

Target User

The Rapid Intelligent Prototyping Laboratory (RIPL) system is a prototype designed to be used by human factors engineers and computer interface designers. In concept, RIPL is designed to allow an interface designer to quickly generate a display prototype while having on-line access to the Smith and Mosier (1984) guidelines (S&M), and an expert system that can "look over the designer's shoulder" while the display is being created. After creating the display it would be fed into an evaluation tool that makes suggestions about possible improvements. In addition, multiple display screens could be linked together and the actual system simulated for user training/evaluation.

Decision-Aiding Style

RIPL consists primarily of 5 design tools:

- A graphics editor to draw items on the screen or import images from Apple Macintosh drawing applications. To generate sample displays, RIPL has 2 graphic editors. The first is a set of drawing tools that can be used to generate primitive objects such as circles, squares, lines, etc. The second program allows the importing of bit-mapped images from the Apple Macintosh. Once the display is generated, the simulator program is used to model the dynamic behavior of the system. It is not clear how the interaction between screens is input, nor how buttons, dialogue boxes, and other screen input is handled.

- A simulator program that simulates the dynamics of the user interface. The simulator program is a set of specifications about tiles (windows) and a set of timers. The timers simulate the system's activity. The user (or designer) can then interact with the "system" by entering events with a keyboard or a pointing device. All of the user inputs are stored in a file and can be played back and analyzed.

- A program that evaluates the display. The Metric calculator is a tool that evaluates displays by checking various parameters. It checks for objective data such as positional consistency, stimulus response consistency, and calculates screen density.

- A technical library of design guidelines. The Technical Librarian is an on-line version of S&M. The program can used to search for display guidelines that include keywords.

- An expert system that helps the designer, through the use of selected design rules, excerpts from S&M, and a dialog system that helps him locate relevant design guidelines. The RIPL Expert System (RIPLES) is a help facility based, in part, on the Smith and Mosier (1984) guidelines as a knowledge base. RIPLES operates on this knowledge base with a rule based, backward chaining inference engine. It uses the EMYCIN style of inferencing and explanation. Basically, the system has several general goal solutions and, through a dialogue with the user, helps to identify the "best" solution.

Architecture

RIPL is a program that runs on a VAX workstation. All of the code is written in Pascal. RIPL's minimum hardware requirements are a VAXstation I or II with 4 meg RAM minimum, a 19" monitor, a pointing device, a keyboard, and a 31Mb hard disk. The environment of choice is a VAXstation II with 9 meg of RAM and a 71Mb hard disk.

Design Heuristics

The heart of the RIPL knowledge and rule-base is the S&M guidelines. These guidelines are used in three different ways.

- First, the complete set of guidelines is available on-line and can be accessed directly or t hough a search facility. This is essentially the technical library.

- Second, a subset of the guidelines are put into an IF-THEN form and this is used in the expert system RIPLES.

- Third, a separate subset of the guidelines is used in the metric calculator to evaluate the display. The specifics of the display are compared against these guidelines and discrepancies are reported to the user.

Evaluation Metrics

The "metric calculator" is the evaluation tool. It checks parameters, such as screen density, location and size of the windows, and stimulus response consistency.

Evaluation

Strengths

1. The RIPL system supports the rapid prototyping paradigm.

2. RIPL includes on-line User System Interface expertise through a proof of concept expert system, evaluation metrics, and an on-line manual service.

Weakness

1. RIPL has never been tested outside the laboratory development environment.

2. The RIPL system does not support the use of independent graphics.

3. Only a small subset of the S&M guidelines are used in RIPL's evaluation.

4. RIPL needs to be optimized to overcome its slow performance issues.

5. The is no provision to allow dynamic additions to the guidelines or constraints used by the metric calculator.

Problem Coverage

In concept, RIPL sets out to provide an impressive array of tools for interface prototyping. The associated documentation, however, is lacking in descriptions of the display and simulation modules. It emphasizes the technical library and consultation program. There are no examples of simulated displays and it is not at all clear how the simulation package works.

The focus of the system is to demonstrate the feasibility of the various design tool concepts that it incorporates. These tools are the on-line Smith and Mosier document and a rule-based expert system. Theses items appear to be the deepest implementations, with less effort having gone into the display editor. It seem that the "Technical library" is simply a text file that can be searched for key words. The problems the expert system supports are very restricted. The "Metric calculator" is an ambitious aspect of the system, but it evaluates a display without a consideration of its purpose. A more important function would be for it to support task-dependent display evaluations.

Current Applications

The Rapid Intelligent Prototyping Laboratory is a prototype system designed to aid in the development of realistic training environments for Air Force applications.

9.7 Foley et al. 1988

Foley, J., Gibbs, C., Kim, W. C., and S. Kovacevic, "A Knowledge-Based User Interface Management System," Proceedings of CHI '88, pp. 67-72.

Description

Target User

The User Interface Design Environment (UIDE) is a system designed to assist in user interface design and implementation. UIDE is intended to free the interface developer from low-level primitive implementation issues to allow him/her to concentrate on the functional issues of interface design.

Decision-Aiding Style

UIDE attempts to go beyond the capabilities of the typical User Interface Management System (UIMS), which requires the designer to work at the syntactic and lexical level of design and, to focus on command names, screen and icon design, menu organization, sequencing rules, and interaction techniques.

Architecture

The heart of the UIDE system is the knowledge base representation of the conceptual design of a user interface. This representation consists of the class hierarchy of objects in the system, properties of the objects, actions which can be performed on the objects, units of information required by the actions, and pre-and post-conditions for the actions.

The knowledge base is used for several purposes: to represent the conceptual design of the user interface; to transform the knowledge base, and hence, the user interface it represents into a different but functionally equivalent interface; and to implement the user interface via a user interface management system.

UIDE is implemented on a SUN 3/160 with ART. Schemata are used for the knowledge base, and rules are used to implement the transformations and UIMS.

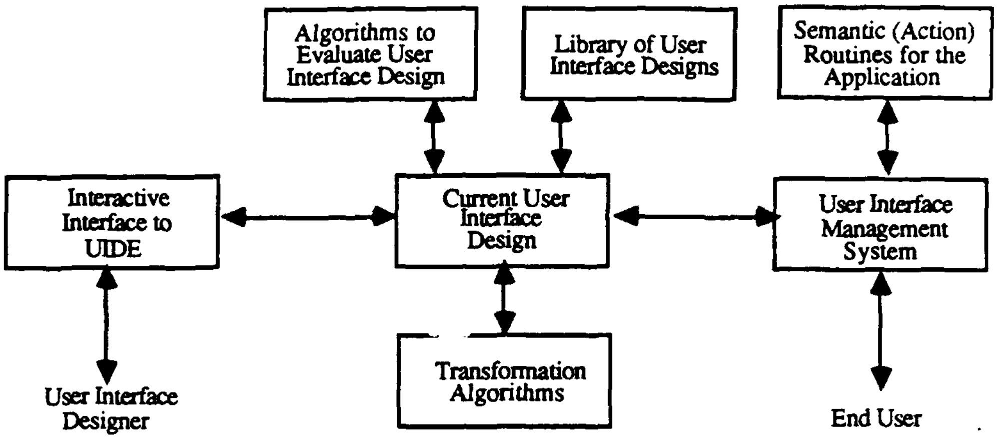

Algorithms to Evaluate User Interface Design
Library of User Interface Designs
Semantic (Action) Routines for the Application
Interactive Interface to UIDE
Current User Interface Design
User Interface Management System
User Interface Designer
Transformation Algorithms
End User

Figure 9.3 The design concept for UIDE.

Design Heuristics

The UIDE system accepts as input the knowledge base and implements the user interface it describes. By applying transformations or changing the knowledge base, the designer can quickly implement a variety of functionally equivalent user interfaces. Also, UIDE can automatically generate a set of alternative designs by applying transformations from the knowledge base. In addition to the knowledge base, UIDE must be given action routines to carry out the application semantics.

The transformations that UIDE uses include factoring, special case object creation, command selection, attribute value selection, set generalization, value default, specialization, and consolidation. The transformations map one user interface into another by operating on the knowledge representation of the interface. Interface transformations allow the system and designer to create families of functionally equivalent user interface designs for one application program.

Evaluation Metrics

The user interface designer uses UIDE to develop alternative designs that can be manually evaluated. Some of the alternatives will be slight variations on one another, while other designs will be quite different. By evaluating different families of interface designs, the developer can test different effects of design strategies.

The basic interface specifications are entered into the knowledge base representations by the interface developer. Once this base design is established, UIDE helps the designer explore alternative ways to structure the semantics of the application.

Evaluation

Strengths

1. The UIDE system supports the rapid prototyping paradigm.
2. UIDE attempts to be a user interface management facility.
3. UIDE supports the automatic generation of variation families of an interface design.

Weakness

1. The UIDE system does not check the interface design for consistency and completeness.
2. Evaluation of an interface design with respect to speed of use and ease of learning is not supported by UIDE.
3. UIDE does not support any run-time help for the user.
4. The UIDE system only supports single inheritance for class objects.
5. Entering the basic interface specifications can be an almost impossible task for the designer.
6. Even though many interfaces can be tested and created, it is possible that the design phase paradigm can get "bogged-down in refinement" on a poor design of an alternative.

Problem Coverage

The knowledge base represents user interface design knowledge at a level of abstraction higher than is typical of other UIMS. In particular, it represents objects, actions, attributes, classes, and conditions. The knowledge base can be algorithmically transformed into a number of functionally equivalent interfaces, each of which are slightly different from the original interface.

The transformations can be used to generate design alternatives. In addition, all interfaces which can be created by applying successive transformations form an equivalence class of interfaces. UIDE has a canonical form representation to characterize each equivalence class in a standard way.

Current Applications

The current implementation of the User Interface Design Environment is still being developed.

9.8 Frey and Wiederholt 1986

Frey, P. R., and B. J. Wiederholt, "KADD - An Environment for Interactive Knowledge Aided Display Design," Proceedings of the 1986 IEEE International Conference on Systems, Man, and Cybernetics, 1986, pp. 1472-1477.

Description

Target User

The Knowledge Aided Display Design (KADD) system is a design environment designed to assist the computer interface developer in the analysis, design, and evaluation of graphical displays.

Decision-Aiding Style

KADD attempts to overcome the limitations of written human factors guidelines by providing a design environment that incorporates on-line human factors knowledge to support a structured approach to display design. The focus of this design environment is not just to present information to the user, but to structure the designers decision making process through a nominal design process model.

Architecture

The KADD system was developed in the Apollo Workstation environment (Aegis). The implementation of KADD consists of four major components:

- The database management system (DMS) used to create and maintain the problem-dependent information requirements database.

- The display editor allows the designer to create and edit display elements to satisfy the end users' information requirements. The display editor is composed of two tools. The first tool is a display element editor which allows the designer to manipulate high-level, domain specific display elements. The second tool is a graphics editor which allows the designer to design static display elements from primitives.

- The display design expert system contains human factors knowledge. This expert subsystem provides evaluation of displays and suggestions for improvements on those displays during the design phase.

- The display driver allows the designer to review the displays in a simulated environment.

Design Heuristics

The KADD system provides an extended design environment, including the tools of the display design trade (e.g., analysis support, data base management, documentation control, graphics editing), on-line guidance related to the current design problems, and access to a variety of previous design solutions.

Evaluation Metrics

The KADD system enforces a nominal design process. A systematic approach to display design is considered necessary to ensure that all the information requirements are met by the displays. By enforcing a structured design process, the system can help the designer satisfy and evaluate the objectives. This nominal design process also allows the designer to simultaneously manage multiple design solutions for future evaluation, in accordance with the constraints of the system.

Evaluation

Strengths

1. KADD is implemented as a design environment rather than as a design consultant or expert.
2. After a design is essentially complete, KADD can assist with the creation of supporting design documentation.
3. KADD is designed to allow human factors knowledge to be used early in the design process when a significant impact may be made on end user workload, efficiency, and effectiveness.

Weakness

1. KADD has difficulty making assessments across elements within a presentation or across display formats.
2. KADD relies heavily on the Apollo environment in which it was developed. Any changes within the Apollo environment will affect the KADD environment, respectively.
3. With every additional feature, KADD increases with overall size and complexity.

Problem Coverage

The first phase in the interface design process is the recording and manipulation of the presentation requirements and their interrelationships within a function and task analysis database. The framework for this recording and manipulation is provided by KADD's information requirements database subsystem. Information requirements and data characteristics form the analysis foundation upon which the presentation interface will be built. The designer is responsible for, but not required to fill out the requirements database before the beginning of the display design.

The next step in the design process is the development of display elements to satisfy the information requirements. The display editor is the primary component of the KADD system used for this purpose. The designer can manipulate any display element attribute within certain human factors limitations. However in the earliest creation stages, particular ranges and precisions are not constrained in order to give the designer more flexibility.

As the designer is creating a presentation interface, human factors encoded rules are applied for checking consistency, requirements satisfaction, and to provide design guidance. Designer specific rules can be added and modified according to new factors and unique situations using a rule editor. The expert system inference engine is constantly background evaluating the design product as the designer is developing it. At any time, the designer can request an evaluation on the current display product.

In the final phase of the interface design process, a dynamic simulation facility is included in KADD to allow the designer to view and subjectively evaluate the interface under a variety of changing situations.

Current Applications

Currently, KADD is being developed to aid in the design of displays for transport aircraft crew stations.

9.9 Friedell 1984

Friedell, M., "Automatic Synthesis of Graphical Object Descriptions," Computer Graphics, 1984, Vol. 18, No. 3, pp.53-62.

Description

Target User

The automatic synthesizing of graphical object descriptions to dynamically present a detailed display to a user in response to a data base query.

Decision-Aiding Style

The system automatically synthesizes graphical object descriptions from high-level specifications through mechanisms for describing, selecting, and combining primitive elements of object descriptions. Underlying these mechanisms are a referential framework for describing information used in the construction of object descriptions and a computational model of the object-synthesis process.

Architecture

Object synthesis proceeds incrementally. The focal point of this process is a data structure called the object frame. Each step in object synthesis is effected by the application of a synthesis operator to the object frame. The choice of which synthesis operator to apply at a given point is based on a computational model of object synthesis called the synthesis agenda. Synthesis control supervises the selection of synthesis operators based on the synthesis agenda and coordinates the overall object synthesis process.

The collection of all synthesis operators is known as the graphics knowledge base. The graphics knowledge base is a formalized body of knowledge pertinent to object synthesis that has been encoded for automatic use by a computer.

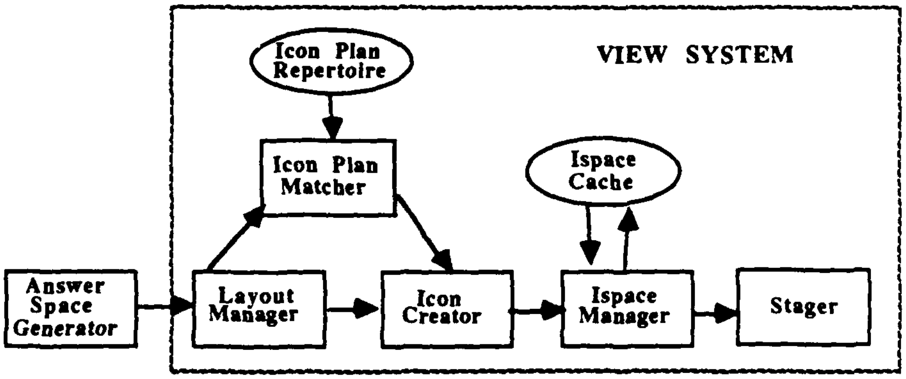


Figure 9.4 The View System's architecture.

The situation space is a performance oriented, knowledge-indexing scheme that is the basis of synthesis operator selection. The structure of the situation space facilitates efficient search for synthesis operators that are the most relevant to the current object synthesis situation.

Design Heuristics

Object descriptions are synthesized by reasoning about how to select and combine widely applicable primitive elements of object descriptions. This allows a broader range of object descriptions to be created. Object synthesis is directed at producing both two-dimensional objects for use in information displays and three-dimensional objects for use in realistic scenes. This synthesis technique is designed for real-time performance.

Evaluation Metrics

By using the graphics knowledge base and situation space, the automatic synthesis of graphical object descriptions determines how to present information without the need for complex evaluation of the presentation format.

Evaluation

Strengths

1. Automatic synthesis of objects can greatly reduce the cost of preparing descriptions of complex scenes.
2. Dynamic synthesis techniques allow the use of computer graphics in situations which predefined graphics could not be provided.

Weakness

1. The automatic synthesis of graphical object descriptions only addresses a part of the overall design issue of user graphical interfaces.
2. There is no provision for user control primitives or interaction primitives within the synthesis technique.
3. Acquiring the specific graphics knowledge to fill the knowledge bases is a difficult task.
4. There is no provision for evaluating graphical layouts.

Problem Coverage

Computer graphicists commonly build object descriptions by decomposition and aggregation. Consequently, it is easier for human computer graphicists to articulate advice for synthesizing subobjects than for complex aggregate objects. To take advantage of this, the object synthesis technique makes use of subobjects.

The graphics knowledge base defines objects in terms of subobjects. Each object is linked through the subobjects relation to the subobjects that compose it. These subobjects are linked by structural relations that define how they are oriented with respect to each other.

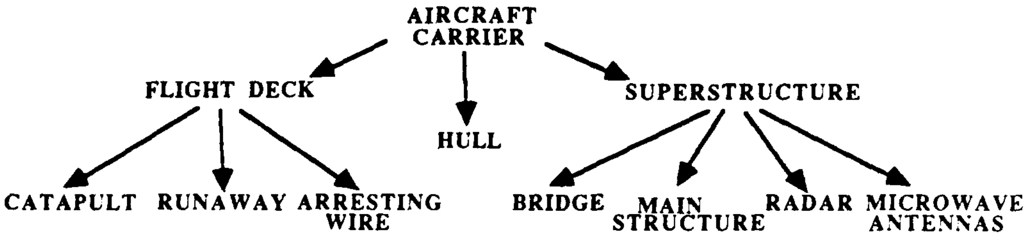


Figure 9.5 A View System object reduction tree.

An interpreter is responsible for accepting descriptions for the aggregate object, generating descriptions for appropriate subobjects, and combining the resulting subobject descriptions into an object description for the aggregate object. The interpreter uses the graphics knowledge base to recursively grow an object-reduction tree in a depth-first fashion. The roots of the reduction tree contain pointers to the parts of the object description. Passing these pointers to a presentation generator results in the appearance of objects on the display.

Current Applications

View System is a graphical user interface to databases that has been used for Naval Applications.

9.10 Gargan et al. 1988

Gargan, R. A., Sullivan, J. W., and S. W. Tyler, "Multimodal Response Planning: An Adaptive Rule Based Approach," Proceedings of CHI '88, pp.229-234.

Description

Target User

The adaptable multimodal response planner is a design for an intelligent interface system that attemps to apply research from perceptual and cognitive psychology, user modeling, and automatic presentation generation to provide a multimodal, response adaptable, intelligent, interface for applications.

Decision-Aiding Style

The adaptable multimodal response planner dynamically determines the appropriate way to present information to the user. The developers believe there are two main advantages to this style. The first advantage is that the presentation can be generated dynamically based upon the current context of the interaction and the current user. The other advantage is the use of knowledge bases to support the presentation of information. Knowledge bases are modular, so they are considered easier to maintain than if the knowledge was embedded in the design.

Architecture

The response planner, which is a component of a larger, previously defined, intelligent interface system, is composed of two phases.

- The Modality Selection Phase determines the presentation modality. It is designed to address the issues of how much information each mode can express, what amount of redundancy in information presentation is optimal for a given presentation, and what combination is optimal for the given user and context.

- The Technique Selection Phase selects one or more techniques within each mode to present information covered by that mode.

Design Heuristics

The adaptable multimodal response planner system utilizes a rule based approach to select one or more techniques to present information within each of the previously selected modalities. The system is designed to adapt to individual users and to provide a level of response flexibility not found in traditional presentation systems. Models are used for storing knowledge about the user. Models are enhanced as new data is obtained and adapted to the needs of the users.

Evaluation Metrics

The adaptable multimodal response planner system utilizes hierarchically structured knowledge stored in four user models. They are the Canonical Model which contains general perceptual information common to all users, the Stereotype Model which contains information about classes of users in a particular domain, and the Individual User Model which contains initially a copy of the stereotype model and is modified over time to reflect the characteristics of the particular user. By using a hierarchical organization of user models, the response planner can utilize information specific to an individual or class of individuals when determining how to present information without the need for complex evaluation of the presentation format. The specific knowledge the system embodies is not described in the paper reviewed.

Evaluation

Strengths

1. The system's model structures are designed to provide a mechanism for allowing the response planner to work in multiple application areas.
2. The use of user models allows the response planner to dynamically plan responses differently to different users.
3. The approach of selecting presentation modality before selecting mode techniques seems valuable from both a time savings and a multimode communication perspective.

Weakness

1. The adaptable multimodal response planner system has not been implemented.
2. Parts of the response planner architecture, such as pictorial and temporal modalities, are not fully expanded.
3. The adaptable multimodal response planner only addresses explicit user intention.

Problem Coverage

In the first phase of response planning, the planner analyzes the information being presented and reasons about how to partition the information and assign it to a particular perceptual modality for presentation. This assignment is based upon a determination of the information/modality pair that supports maximal understanding and also accounts for the load placed on each modality by the current interactive context. The modality selection phase is subdivided into three steps - expressiveness, effectiveness, and adaptation. The first step determines which modalities are able to present the information. The second step determines what information should be presented in more that one modality in order to maximize understandability. Finally, the third step accounts for any difference based on the user cognitive biases and current context. At the end of this step, the response planner has determined the range of information covered by each mode.

Once the information has been assigned to the individual modalities, the next task is to reason dynamically about which technique can best express the information within a particular modality. The technique selection phase is also subdivided into three steps - expressiveness, effectiveness, and adaptation. First, the response planner determines which techniques are able to present the information within a particular modality. Second, it will choose one or more techniques which present the information most clearly based on perceptual clarity and understandability. Finally, the response planner accesses hierarchically structured knowledge about the user from user models. After this final stage, the presentations are rendered.

Current Applications

The adaptable multimodal response planner is a design for a system that has been applied to examples from the manufacturing domain.

9.11 Hayes 1985

Hayes, P.J., "Executable Interface Definitions Using Form-Based Interface Abstractions," in H.R. Hartson (Ed.), Advances in Human Computer Interaction, Ablex Publishing Co.: Norwood NJ, 1985, pp.243-281.

Description

Target User

Cousin-Spice (C-S) is a high level programming language for creating alphanumeric displays. It is designed for use by a human factors engineer to prototype displays. The C-S language does not appear to be very complex, although to use it probably requires some programming skill.

Decision-Aiding Style

Hayes makes three proposals for interface design: Standard interface primitives, separation of interface and application, and form-based metaphors.

Hayes introduced the form-based abstractions, or the idea that the interface could be like a form that the user fills out and then sends to the application. This form can be used for editing and changing the parameter values, jumping from field to field at will, selecting pre-determined values from a list (menu), having default values appear automatically (while still editable), and prohibiting the user from entering non-applicable data. The application does not interact with the form until a request is made by the pressing of a "button."

Forms are designed to provide:

- Correction of erroneous or abbreviate input - This insures that the data input is the right type and within an acceptable range.
- Interactive error resolution - This provides help to the user by displaying all of the possible options for a given parameter.
- Integral on-line help - This is, in part, built in to the form abstraction, providing the names of input fields. In addition, the user can request additional information regarding the field formats and boundaries.
- Automatically generated on-line documentation - The system can provide information to the user about the purpose for each field and it correct usage.

The tool developed to use this language for the production of interfaces is called Cousin-Spice. C-S is an interface definition language for the creation of buttons and fields with defining and constraining attributes. The features that C-S provides are:

- editable fields
- items choosable with a pointing device
- buttons
- pop-up menus
- immediate feedback regarding incorrect or missing input values

Cousin-Spice is an interpreter that takes a text input file (much like a program) and produces a screen with editable text fields, buttons, and menus (lists of available options). This screen (or "form" as it is called) is simply the interface for an application. Cousin-Spice does not affect the application, but simply provides data and parameters that the applications requires. The user can then fill out the "form" and press a button to have the application perform an action.

The use of *transition networks in the design of* displays was discussed. Hayes feels that a transition network is a good model for Teletype terminals (because of their finite-state nature) but not for graphical interfaces (because of their infinite-state nature and the need for interface changes where no mode change exists). Hayes suggests that the form-based abstraction is better suited for graphical interfaces than are transition networks or context-free language models.

Architecture

Cousin-Spice is an interpreter that runs on a Perq workstation. The Perq workstation features bit-mapped graphics and 1MB of memory.

Design Heuristics

No interface design heuristics given.

Evaluation Metrics

None given.

Problem Coverage

Supports the creation of executable form based interfaces through a modified state transition dialogue language.

9.12 Lockheed Corporation 1987

Lockheed Corporation, Lockheed User Interface System (LUIS) User Manual, Austin TX: Lockheed Austin Division, December 1987.

Description

Target User

LUIS is a User Interface Management System (UIMS) that assists the computer systems developer by providing high-level and flexible mechanisms for the creation of functional user interfaces. A set of graphical objects that serve as building blocks for constructing different user interfaces is provided by the system. With LUIS, the developer can use high-level mechanisms on any object to produce instances with new attribute specifications. Any interface created with LUIS is based upon some combination of these primitive objects.

Decision-Aiding Style

The LUIS System is a software package that supports the rapid prototyping and management of user interfaces. Interfaces can be developed quickly to make early, yet rigorous evaluations of candidate designs possible. In its interface management role, LUIS supports not only the creation and control of displays but also the integration of the user interface within a system of applications.

Architecture

LUIS is implemented in "C" and has been run on Apollo, Sun, and Silicon Graphics workstations in UNIX and Aegis environments. Therefore, LUIS interfaces can be used for applications that reside in the UNIX or Aegis environment that are not necessarily written in the C language.

The LUIS system, itself, executes as a self-contained computer program. The specification of a particular user interface serves as input into LUIS. The program processes the input and produces an interface that executes in real-time. Specifying the user interface is accomplished through either a high-level textual command language or a menu-based editor.

LUIS consists of four subsystems that carry out basic functions that support the construction and management of user interfaces. These are:

- A parser that process a high-level textual description of the objects comprising a design.

- An interactive dialog editor that provides menus and keystroke commands for the creation and modification of interface objects.
- An interpreter that executes the specified user interface. The interpreter processes end-user inputs, manages LUIS objects, and invokes the application routines.
- A graphics editor that supports the interactive creation and tailoring of graphic objects for use in LUIS user interfaces. The graphics editor contains features that directly facilitate the construction of icons and analog objects.

Design Heuristics

In the design phase of a LUIS interface, the implementers rely on an independent specification of the requirements for the interface at hand. This specification is usually obtained by a Human Factors Engineer who has observed users of an existing system or polled potential users of the newly conceived system. The implementers, using the specification, produce prototype interfaces for the computer system.

Evaluation Metrics

Evaluation of a LUIS interface is done manually in a two step fashion. The first step is to allow the Human Factors Engineer to critique the interface against his specification. Once the interface is approved by the Human Factors Engineer, the second phase is to test the interface on a small group of potential users.

Evaluation

Strengths

- LUIS supports the rapid prototyping paradigm.
- User interfaces can be created without a driving application system.
- LUIS attempts to be a user interface management facility.

Weakness

- No empirical schema for interface design heuristics or evaluation metrics.
- LUIS has difficulties regulating the interaction between the application's control and the control of the user interface.
- Only objects that are in the LUIS graphics editor database can be use to construct user interfaces. Any other primitive objects must be painstakingly designed and added to that database.

- Even though many LUIS interfaces can be tested in parallel, it is possible that the design phase in the LUIS paradigm can get "bogged-down in refinement" on a poor design of an interface candidate.

Problem Coverage

Once the specification has been defined for a user interface by a Human Factors Engineer (HFE), the implementers create several candidates for the interface using the LUIS system. All candidates are designed within the specification limits, but can differ across implementers' interpretations.

Once the candidate designs are finished, they are presented to the HFE for evaluation. The HFE critiques each interface against the written specification. The candidates that pass this evaluation are modified by the implementers according to any recommendations provided by the HFE. This is an iterative design process that may be repeated numerous times before an interface meets the HFE specifications and passes his evaluation.

In the next phase of the LUIS interface construction, the HFE tests the interface on a representative set of typical users of the application. The HFE observes how the interface is being used and what modifications are needed. The users critique the interface to the HFE. If the interface needs more modifications or, possibly a total revision, the HFE may take the candidate interface back to the previous phase.

In the final phase, the interface is integrated with the application program(s). The HFE must critique how the interface performs in conjunction with the application. Also, a sample of users must critique the performance of the system as a whole. Anytime the interface does not meet the pre-determined standards, it is sent back one phase in its development plan.

Current Applications

SMS -- Lockheed 's Softcopy Map Display System, among others.

9.13 Mackinlay 1986

Mackinlay, J., "Automating the Design of Graphical Presentations of Relational Information," ACM Transactions on Graphics, 1986, Vol. 5, No. 2, pp. 110-141.

Description

Target User

A Presentation Tool (APT) was developed as an application-independent system that automatically designs graphical presentations of relational information. Using APT, application designers do not predesign the presentations, but default the responsibility of graphic design to APT.

Decision-Aiding Style

The APT approach is based on the design concept that graphical presentations can be described as sentences of graphical languages. Also, the APT approach uses a graphical design as an abstract description of an image that indicates the graphical techniques that are used to encode information. The graphical design issues are codified as expressiveness and effectiveness criteria for graphical languages. Expressiveness criteria determine whether a graphical language can express the desired information. Effectiveness criteria determine whether a graphical language exploits the capabilities of the output medium and the human visual system. A wide variety of designs are systematically generated using a composition algebra, which is a collection of primitive graphical languages and composition operators that can form complex presentations.

Architecture

APT combines a synthesis algorithm that generates designs in order of effectiveness criteria. The synthesis algorithm consists of a design component followed by a rendering component. The design component uses logic programming techniques to implement the synthesis algorithm. The rendering component use object-oriented programming techniques and a device-independent graphics package to render the resulting designs. APT's synthesis algorithm is based on a divide-and-conquer search strategy. When a particular set of choices does not lead to a composite design, backtracking is used to consider other choices. APT uses a depth-first search with simple backtracking.

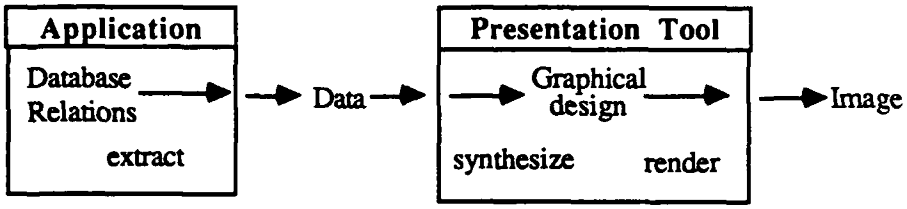


Figure 9.6 The APT process of automating graphic design.

The APT system was developed on a Symbolics LISP Machine using MRS, a representation and logic programming system. APT is a prototype, and no effort was made to make it efficient. The logic program is made of approximately 200 rules, and the rendering system is about 4000 lines of Lisp code.

Design Heuristics

The naive approach to generating graphical design alternatives is to develop an ad hoc list of graphical languages that can be filtered with the expressiveness criteria and ordered with the effectiveness criteria for each input. The major difficulty with this approach is that there is no guarantee that there will exist an appropriate design. In APT, a composition algebra is used for generating graphical design alternatives. Such an algebra consists of a basic set containing primitive graphical languages and some composition operators that can generate composite designs for evaluation.

Evaluation Metrics

Expressiveness and effectiveness criteria are used, by APT, to evaluate graphical designs. Expressiveness is based on the premise that universal graphical conventions indicate arrangements of graphical objects that can be used to encode information. Effectiveness is based on the premise that users accomplish perceptual tasks associated with the interpretation of graphical presentations with different degrees of accuracy. The expressiveness criteria is used to encode the syntax and semantics of a graphical design. The effectiveness criteria are used to rank order the candidate designs.

Evaluation

Strengths

1. APT attempts to formalize a graphic presentation as a collection of graphical languages.
2. The APT system attempts to automate the graphical presentation design process.

Weakness

1. The focus of APT is two dimensional static presentations.
2. The rendering component of APT places an implementation restriction on the set of primitive languages that does not allow orientation, texturing, map displays, and line charts.
3. APT is not an efficient implementation.
4. Dealing with multiple and conflicting effectiveness criteria is beyond the scope of APT.

Problem Coverage

The fundamental assumption of the approach that APT uses is that graphical presentations are sentences of graphical languages, which are similar to other formal languages in that they have precise syntactic and semantic definitions.

An expressiveness criterion, which is derived from a precise language definition, is associated with each graphical language. A graphical language can be used to present some information when it includes a graphical sentence that expresses exactly the input information. Expressing additional information is potentially dangerous because it may not be correct.

Given the focus on accuracy, effectiveness criteria are based on the observation that a graphical language uses specific graphical techniques to encode information. When interpreting a graphical sentence, a user is confronted with perceptual tasks that correspond to these graphical coding techniques. Since some perceptual tasks are accomplished more accurately than others, effectiveness criteria can be based on the comparison of the perceptual tasks required by alternative graphics.

Since most graphical presentations of relational information are based on a general vocabulary of graphical techniques, a variety of design can be generated with a composition algebra the describes this graphical vocabulary. The composition algebra consists of a basis set that contains primitive graphical languages, each embodying one of the graphical techniques for encoding information, and composition operators that are able to generate a wide range of presentations by composing the primitive languages.

Current Applications

The APT system has been applied to a limited set of research examples.

9.14 Tullis 1986

Tullis, T. S., "A System for Evaluating Screen Formats," Proceedings of the 30th Annual Meeting of the Human Factors Society, 1986. (Display Analysis Program (version 4.0). Lawrence Kansas: The Report Store 1986).

Description

Target User

The Display Analysis Program was developed to measure the format characteristics of alphanumeric displays and to suggest improvements for a designer to implement.

Decision-Aiding Style

When analyzing a presentation format, The Display Analysis Program outputs as its final summary, a sequence of short messages that consist of the results of the evaluation and suggestions for improving the screen display. It also provides interpretive displays that relate to each of the evaluation criteria.

Architecture

The Display Analysis Program, that runs on an IBM-PC, is available through the Report Store. The system evaluates pre-existing alphanumeric screen layouts input as ASCII files against six criteria and suggests changes when associated parameter values are exceeded. It also predicts visual search time and subjective ratings using a regression model with parameters estimated from a sample of 520 displays.

Design Heuristics

The Display Analysis Program cannot be used as a total design environment, but only as an automatic evaluation tool.

Evaluation Metrics

1. Overall Density of Characters on the screen - Measures percentage of total screen spaces filled by characters, as well percentages of uppercase, lowercase, digits, and others. If % uppercase > 50, a warning is issued with respect to readability (no justification for 50% criteria given). If overall density > 40%, a recommendation to reduce screen content is printed (the selection of the density parameter that triggers the warning is based on a review of 520 displays (Tullis 1984) - 40% is one standard deviation above the mean for that sample).

2. Local Density - Measures the percentage of character spaces within 5 degrees visual angle of each character that hold other characters. Results are shown in terms of a display density map coded by ten symbols, with the location and density of the highest density character indicated and average local density. If average local density is > 62% a message is printed indicating high density and a recommendation that the designer try to "spread out" the characters.

3. Number of Visual Groups - Uses a proximity clustering technique to predict the "visual groups" that a user would perceive. Outputs are a "group map" that codes each group with a different character, and statistics summarizing the number characters in each group.

4. Average Visual Angle Subtended by Each Group - Lists the visual angle in degrees, that each visual group subtends. This is followed by maximum visual angle subtended by a group, and the mean visual angle subtended for all groups, weighted by the the number of characters in each group. If one of the groups subtends a visual angle >13 degrees, a message is printed, pointing out that the display contains large groups with a suggestion that the existing groups be decomposed further. The message suggests how this can be done to insure that the user will perceive the components as distinct groups. If the number of groups in the display is >50, a message is printed explaining that the the display contains many groups, and suggests that they be combined into fewer, larger, groups. These rules were derived from search time research (Tullis, 1986) that suggests the optimum trade-off between the number and size of groups falls between 19 and 40 groups with an average size of 2.4 to 4.9 degrees of visual angle.

5. Layout Complexity - Generates a "layout map" in which a single character is used to represent the position of each "item" on the screen: a "C" for the starting position of each character item, and an "N" for the actual or assumed decimal point of each numeric item. The program also computes a measure of display layout complexity (Tullis 1983, 1984) by counting the number of different rows and columns used. This measure attempts to assess the amount of information conveyed by the positions of items in the display to give the designer a sense of how well-aligned the items are with respect to each other. Tullis (1984) found that layout complexity was the best single predictor of subjective display evaluations such that the higher the complexity rating, the worse the evaluation. If the total complexity of the display is > 8 bits, a message is printed indicating that the display complexity is high and suggests that numeric items be vertically aligned on their decimal points, and that character items be vertically aligned on their starting characters.

6. Search time and evaluation prediction - Using the values estimated for six aspects of the display, and the regression models developed by Tullis (1984), the program prints out the estimated time, in seconds, that it would take someone to locate an item in the display and the predicted subjective user rating of the display's "ease of use." Tullis reports that the search time regression equation has typically explained from 58 to 96 percent of the variance in actual search times, but that the actual predictions are normally lower than the observed search times. As a consequence, the recommended interpretation of these predictions is for comparisons of alternative displays of the same basic information.

Evaluation

Strengths

1. The Display Analysis Program is a utility that can be used to study different display parameters in different task domains.
2. Six objective evaluations are made by the Display Analysis program and it provides a unique set of interpreted displays to help the designer improve a design with respect to these criteria.

Weakness

1. The Display Analysis Program only works on static alphanumeric displays.
2. Display Analysis is only a tool for evaluation of static screens and, therefore, needs to be coupled with a design environment.
3. The evaluations made by the Display Analysis Program are content independent.

Problem Coverage

Display Analysis accepts as its input an ASCII file containing a literal example of the screen to be analyzed. The program's output is a five page printout, including graphics, formatted in terms of the six evaluation criteria. On the first page, the program prints the display exactly as it was read from the file. The second page presents the results of the local density analysis. The third page contains the results of the grouping analysis. The fourth page consists of the results of a layout analysis. Finally, the last page contains the values for the six key parameters presented in tabular format, followed by predictions for search time and subjective rating.

Current Applications

The Display Analysis Program is used as a tool for research in human-computer interaction.

9.15 Tyler 1988

Tyler, S. W., "SAUCI: A Knowledge-based Interface Architecture," Proceedings of CHI '88, pp.235-240.

Description

Target User

The Self-Adaptive User-Computer Interface (SAUCI) prototype is a design for an intelligent interface to dynamically tailor a user-computer dialogue to the ongoing context of an interaction session.

Decision-Aiding Style

SAUCI is designed to embody automatic adaptability to changing user demands. The SAUCI developers believe that the increasing range of computer users makes the design of one interface to meet the needs of all system users impractical. Instead, it is considered desirable for the computer system itself to adapt its behavior to the capabilities of its individual users.

Architecture

The interface architecture of SAUCI is viewed as a combination of object-oriented and rule-oriented paradigms. The rules adapt the interface to the context of various knowledge bases which describe the user, the target system commands, and high-level tasks supported by the system. Within this architecture, the interaction of a user with a target system is regarded as consisting of a series of interaction events. Each event is interpreted in terms of several distinct phases. As the user interacts, with the target system, the user moves in sequence through the dialogue phases.

In a general case, three steps are defined for a specific phase. First, the method for the phase activates an appropriate rule set. The rule set then operates upon the available knowledge bases to determine the values of certain parameters that are used by that phase in its operation. Second, the method for that phase then alters the interface by constructing the user-visible interface features dictated by the nature of the phase, the value of the relevant parameters as established by the rule set, and the state of the involved knowledge bases. Finally, the user's behavior during that phase is monitored and incorporated into the corresponding knowledge base.

There are four knowledge bases within SAUCI. The main knowledge base expresses what the system knows about an individual user. A separate object is created for each user of the system. Another knowledge base records, in declarative form, information about the functionality of the system. This is done by representing each command in the target system with a separate object. A third knowledge base embodies the interface's knowledge of the high-level tasks which can be executed on the target system. This information permits the interface to guide the user in executing desired user tasks. The last knowledge base encodes information about the main objects of the target system in terms of the high-level tasks known about the interface. This information is used to provide the user with the context of the interaction in terms of the current state of the world and of those objects being used in the task.

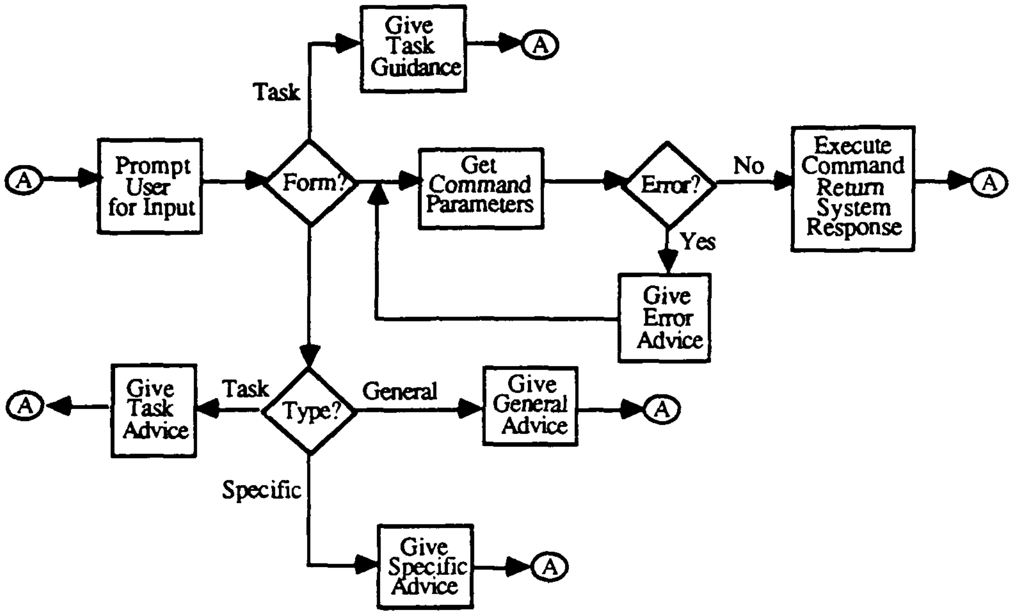


Figure 9.7 Flow-chart of a SAUCI interaction event.

SAUCI employs modularity which means that the interface is embodied in a module separate from the target program. This allows modifying the interface or the program more easily, encourages interface uniformity, and may allow development of general interfaces by specialists that can tie to more than one application.

The SAUCI prototype was written in LOOPS and INTERLISP-D for the Xerox D-machines.

Design Heuristics

SAUCI is not a design environment, it is an intelligent interface.

Evaluation Metrics

none

Evaluation

Strengths

1. The Self-Adaptive User-Computer Interface is a design for an architecture framework to explore issues of interface architecture, knowledge base representation for interfaces, and intelligent interface design and construction.
2. SAUCI includes an intelligent advising facility.
3. Adaptability to the individual user is SAUCI's emphasis.

Weakness

1. SAUCI can only address explicit user intention.
2. There is no implicit schema for evaluation within SAUCI.
3. Acquiring the specific knowledge to fill the SAUCI knowledge bases is a difficult task.
4. The use of graphical or pictorial dialogues are not supported in SAUCI.

Problem Coverage

SAUCI is a general architecture for a run time intelligent interface.

Current Applications

The current implemented SAUCI prototype consists of a working interface to a UNIX operating system on a VAX-11/780.

9.16 Virtual Prototypes Inc. 1987

Virtual Prototypes Inc., Virtual Prototyping Technology Information Package, 1987, Virtual Prototypes Inc.

Description

Target User

Virtual Prototyping System (VAPS) is a tool to be used by system and interface designers. It is designed to allow non-programmers to generate functional prototypes of a display screen in a very short amount of time. The system can simulate most of the user interface, including buttons and switches. The display can be linked to a model of a system (or the system itself) and used as its controls.

Decision-Aiding Style

VAPS embodies the style of the virtual prototype that approximates a real control and presentation system. The virtual prototype will produce displays similar to those required by the target system, to emulate their dynamic behavior. This is because the virtual prototype will incorporate a significant portion of the software required for the real system.

Architecture

VAPS is a software package hosted ‹n a Silicon Graphics high-resolution workstation. The basic VAPS configuration consists of four modules: a Graphics Editor, a Logic Editor, an Integration Editor, and a Run Time Environment. The user will normally interact with all three modules in order to build a prototype. VAPS features a window and menu-based interface, with a mouse serving as the primary interaction devices.

The Graphics Editor is used to "virtualize" hardware, and to build the displays required by the target system. The Logic Editor prepares the control logic of the prototype. VAPS will automatically generate most of the software required to control the interactions. The controls and the displays of the prototype are integrated with a real-time simulation or real data using the Integration Editor. The integration is pictorial and consists of visually connecting sources of data with the objects affected by them. Using the Run Time Environment, the user can interact with the simulated behavior prototype as he/she would with the actual system.

Once the display is generated and optimized there is an automatic code generator that assists in the generation of embedded software for the target system.

Design Heuristics

Designers of VAPS interfaces relay on the manual specification of the user interface requirements. It is assumed that this specification will be developed by a human factors engineer who has observed users of an existing system or has researched and polled potential users of the prototype system.

Evaluation Metrics

A virtual prototype of a target system leads to the "tangibility" of its design concepts to evaluate the credibility of the approach. Evaluation of VAPS interfaces are done manually by observing the interaction of users with a virtual prototype. Step-wise refinement is used to adjust the interface to the needs of the users.

Evaluation

Strengths

1. VAPS has an automatic interface code generation capability.
2. The VAPS system supports the rapid prototyping paradigm.
3. VAPS attempts to be an user interface management facility.

Weakness

1. No empirical schema for interface design heuristics or evaluation metrics.
2. VAPS has difficulties regulating the interaction between the application's control and the user's control of the interface.
3. It is possible in the design phase of a VAPS interface to get "bogged-down in refinement" on a poor design of an interface.

Problem Coverage

VAPS is intended to accelerate the display design process by allowing virtual displays to be created, edited and tested very quickly. Not only are the visual components simulated (dials, gauges, out-of-window views) but physical components can be simulated (virtual buttons and switches provided via touch screen) or directly connected (trackballs, joysticks, steering wheels). In addition, the "invisible interface" i.e. the man-machine interface logic can be programmed into the display. This is addressed by an editor that reacts to input stimuli, adjusts the display parameters, and otherwise makes the prototype a dynamic simulation. In addition, the display can be used to operate actual systems.

Current Applications

VAPS has been applied to aircraft applications such as the design of cockpits, head-up displays, and avionics systems. Also, VAPS has been used in the simulation of surface and underwater combat, the design of operator interfaces in ASW applications, and the presentation and dynamic update of C3I displays on a background of digital terrain.

9.17 Wasserman and Shewmake 1985

Wasserman, A.I. and Shewmake, D.T. "The Role of Prototypes in the User Software Engineering (USE) Methodology," in H.R. Hartson (Ed.), Advances in Human Computer Interaction, Ablex Publishing Co.: Norwood NJ, 1985, pp 191-210.

Description

Target User

RAPID/USE is a programming language that allows a programmer to generate a set of sample display screens (strictly alpha-numeric) and create links between the screens. Thus, human factor engineers benefit the most from RAPID/USE, but the language does not appear simple enough for the casual computer user to program.

Decision-Aiding Style

RAPID/USE is based upon transition diagrams, which are like flow charts where nodes represent states of the system and the links represent actions between states. Transition diagrams can be used to describe the flow of control through a program or process. Wasserman and Shewmake found that, although users could understand the diagrams, diagrams did not provide the user with a true picture of the display. Often, users were not satisfied with interfaces derived from apparently satisfactory diagram specifications. To address this problem, a "programming language" (RAPID/USE) was written to create mock-ups of display screens. The language provided a way to position text at various positions on the screen, and then to edit/revise the text without recompiling the display program. The displays were alphanumeric and keyboard driven.

To simulate the interaction between the presentations (or nodes) a "Transition Diagram Interpreter" was written. The TDI allowed the user to interact with the virtual program and have the program "respond" with the appropriate transition. The resulting system is a virtual program that looks and acts like a real program.

Finally an "Action Linker" was written to provide a more realistic model of the actual program by displaying more complete messages and providing the user more freedom in moving from node to node. In addition, the inclusion of actual program functions allows the virtual program to perform some or all of the actions of the real program. The included functions are accessed like subroutines.

Architecture

RAPID/USE is a programming language that runs on a Sun workstation.

Design Heuristics

RAPID/USE is based on the User Software Engineering (USE) development philosophy and method. The USE development method is comprised of the following steps:

1. Requirements analysis - activity and data modelling leading to preliminary informal specifications, identification of user characteristics.
2. External design - specification of user/program dialogue and interfaces.
3. Creation of a prototype of the user/program dialogue with revisions as needed.
4. Completion of the informal functional specification of the system operations using narrative text.
5. Preliminary relational database design.
6. Creation of a functional prototype system, providing at least some, and possibly all, of the system's functions.
7. Formal specification of the system operations using behavioral abstraction.
8. System design at the architectural and module levels.
9. Implementation in PLAIN.
10. Testing and/or verification.

[Note that the creation of the interface precedes the design of the database.]

Evaluation Metrics

To evaluate a display prototype, all activity is recorded in two logs: a raw input log and a transition log. The raw input log stores all keyboard activity and can be used to count keystrokes, typos, backspaces, etc. The transition log is used to study the movement between nodes. The transition between nodes is recorded and time-stamped. These activity logs can be studied and reviewed to investigate the nature and frequency of user errors

Problem Coverage

Wasserman and Shewmake seem to approach the interface development problem strictly from an alphanumeric display point of view. They feel that the design of an interactive database should start with the user's display screen. The tool that they developed is limited alphanumeric interfaces, but the ideas that they present appear applicable to all displays. These ideas are:

1. Moving toward displays that are command oriented, menu driven, or use natural language.

2. The user interface is often the dominant factor in determining user satisfaction with the system.

3. There is variability in user preference.

4. It is often desirable to have several different interfaces for the same program operation (for different languages or skill levels).

9.18 Weitzman 1986

Weitzman, L., Designer: A Knowledge-based Graphic Design Assistant, Institute for Cognitive Science Report ICS 8609, University of California: San Diego, July 1986.

Description

Target User

Steamer is a computer-based training system to support instruction in the domain of propulsion engineering. It is used to help students develop an understanding of a steam propulsion plant. Designer is a tool to aid users of the Steamer's Graphics Editor by supplementing the programmer's domain knowledge with the graphics expertise.

Decision-Aiding Style

In Designer, a design is sensitive to the context in which it was created. It is this context that defines the external constraints which shape and guide the final solution. Designer refers to context as "style" and is constructed by selecting those constraints that are to be enforced within that "style."

Within Designer, a style is defined by the visual techniques employed in the communication of information. Visual techniques represent a vocabulary in which to describe a design. Techniques, in conjunction with constraints enable the system to suggest a variety of graphics procedures to modify an alternative design. It is the constraints that indicate a discrepancy in the design, while the techniques suggest the graphic procedures that will improve a designer's design.

Architecture

The Designer system consists of three interrelated processes, an Analyzer, a Critiquer, and a Synthesizer, coupled to a domain-dependent knowledge base. This knowledge base consists of design elements, design relationships, techniques for their identification, sets of constraints for establishing a context or style for critiquing a design, and generative techniques for creating design alternatives.

The Analyzer parses a design based on the elements and relationships of the domain, and records this information in the knowledge base. The Critiquer uses the information prepared by the Analyzer along with the domain-based design constraints to indicate where a design succeeds or fails. Using knowledge representing the current state of the design, the Synthesizer generates design alternatives consistent with a design style.

The separation of these three processes from the knowledge base provides the system with a degree of modularity This modularity creates a technology that is easily extensible to other design domains.

In order to support the internal mechanisms of Designer, a series of generic subsystems have been incorporated into the system architecture. These subsystems include Steamer's frame-based knowledge representation facility (MSG) for storing all domain knowledge, and an assumption-based truth maintenance system (ATMS) for maintaining alternative design decisions which define the design space.

Designer was developed in the object-oriented programming environment of Flavors on the Symbolics 3600 family processor.

Design Heuristics

Designer parses a design and locates existing domain elements and relationships. Identifying the elements is straightforward because of the use of predefined icons in the Steamer interface and the object-oriented nature of their implementation. Once the domain elements have been created, Designer locates instances of domain relationships.

Evaluation Metrics

Designer creates an evaluation comment for each unsatisfied design constraint within the current style. Descriptions and justifications of the comments are based on the constraint. Comments are presented to the user and are described in terms of their underlying principle or standard. Thus, it becomes possible to request multiple evaluations based upon different styles.

Design decisions are made to incrementally refine the elements and their relationships. Knowledge of the elements and their relationships along with the comments from the critiquing phase form the basis for design modifications.

Alternative designs are maintained and explored with Designer's ATMS. The ATMS can manipulate justifications and assumption sets. As a result, inconsistent information can exist and working in the problem space becomes effective and efficient. Context switching of alternative designs is supported, and most backtracking and all retraction is avoided. In Designer, the assumptions that are manipulated are the alternatives created by incremental design decisions. Solutions at any stage in the design process are the consistent, noncontradictory environments maintained by the ATMS. Any contradictions that arise are handled by the ATMS and will not appear in the same environment. Based on these assumptions and justifications, an alternative design can describe its derivation and individual decisions can be described in terms of their potential contribution to a final solution. The system conveys design precepts while the user is viewing a specific design alternative. Thus the user's knowledge of constructing visual presentations is enhanced for future design.

Evaluation

Strengths

1. The multiple evaluation paradigm is especially powerful for displays that may need to be presented in different media, each with different constraints.
2. Designer can evaluate existing designs of Steamer presentation displays.
3. Designer responds to the user actions, by analyzing, critiquing, and interactively suggesting improvements to them.

Weakness

1. Designer can not create a new design from high-level design goals.
2. The perception of the problem and the shape of the solution design are both affected by the depth and range of the domain knowledge base.
3. The Designer system needs large amounts of memory to work because of its ATMS environmental storage scheme.

Problem Coverage

In Designer, design involves a cycle of gathering information, making decisions based on that information, and reviewing the consequences of those decisions. New information taken from this process is incorporated back into the cycle for subsequent refinement of the design. This analysis-synthesis-review cycle is the general design process supported by Designer.

In order for Designer to carry out the analysis phase of the process, two steps must take place. The first step is that the system must parse the design into domain elements and relationships. The second step is that the system must locate areas that need to be improved. Together, these processes represent the analysis phase of the design process. Once the first two steps have occurred, Designer is ready to suggest alternative procedures for modifying the design.

Since the overall goal of Designer is to be an on-line assistant and not assume control, review occurs interactively with the user selecting and confirming decisions presented by the system. Information is incorporated back through the process as output from one cycle becomes the input for the next cycle of critique.

Current Applications

Steamer - A computer-based training system for the steam propulsion domain.

10. Appendix C-Commercial Interface Development Products

ADM - Apollo Software, Chelmsford MA.

BLOX - Rubel Software, Cambridge MA.

DATAVIEWS - V. I. Corporation, Amherst MA

DEMO PROGRAM - Software Garden, Newton MA.

ENTER/ACT - Precision Visuals, Boulder CO.

FRAME-UP - Symbolics, Inc., Cambridge MA.

RAPID/USE - Interactive Development Environments, San Francisco CA.

TRILLIUM - Artificial Intelligence Ltd., London, UK.

VAPS - Virtual Prototypes Inc. Montreal, Quebec, Canada.